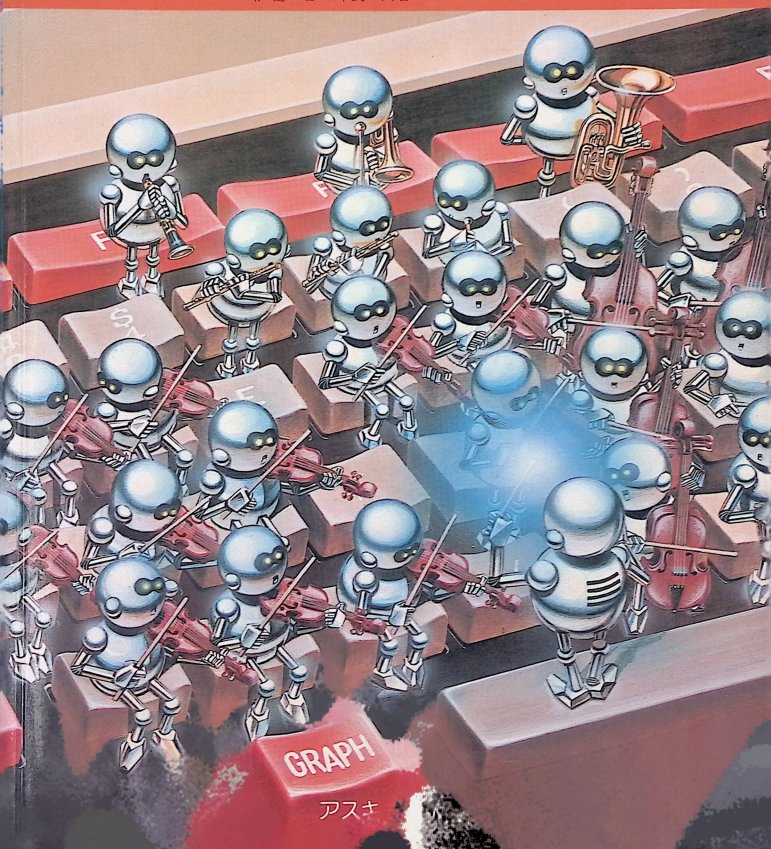
ASCII SYSTEMSOFT

PCファミリー・テクニカル・ノウハウ集 PC-6000シリーズ編第1巻

PC-Techknow6000 Vol.1

一零・樋口理・八木良一共著 システムソフト監修

アスキ

アスキー・システムソフトシリーズ

ＰＣファミリー・テクニカル・ノウハウ集

PC-TechKnow6000 Vol.1

共著／一　零　　樋口　理

八木良一

監修／システムソフト

アスキー・システムソフトシリーズ

PCファミリー・テクニカル・ノウハウ集の発刊にあたって

株式会社システムソフト福岡では，株式会社アスキー出版と共同で，「アスキー・システムソフトシリーズ，PCファミリー・テクニカル・ノウハウ集」を発刊することになりました．

この，「PC-Techknow シリーズ」は，NEC のパーソナルコンピュータ，PC ファミリー（PC-8001，PC-6001，PC-8801）および，その周辺機器を徹底的に活用するためのテクニカル・ノウハウ（Techknow テクノウ）をまとめたもので，構成は以下のようになっています．

- ○PC-8000シリーズ編（N-BASIC）
- ○PC-6000シリーズ編（N_{60}-BASIC）
- ○PC-8800シリーズ編（N_{88}-BASIC）

各シリーズ共，２巻程度にまとめる予定です．

内容は，それぞれの BASIC の内部構造から，キー入力の仕方，カセットおよびディスクファイルの上手な使用法，BASIC プログラム・テクニック，さらには，機械語理解のポイントまで，活用いただける情報が満載されています．

本書は，PC-6000 シリーズ第１巻の「PC-Techknow6000 Vol.1」であり，２巻以後も順次発刊の予定です．

プログラム入力時の注意．

本書に掲載されているBASICプロダラム中，特に断りのないものは，MP-80を使用して出力したもので，CRT画面や，他のプリンターで打ち出したものとは出力形式が若干異っています．

特に，１行が２段にわたるプログラムを入力する際，行番号の下の空白はつめて打ち込んで下さい．

はじめに

半導体技術の急速な進歩は，低価格のパーソナルコンピュータを生み出し，私たちの日常生活にまでかかわりをもつようになってきています．現在，各メーカーからいろいろな機種が発売されていますが，その中でも日本電気株式会社の PC-6000，PC-8000，PC-8800 の 3 シリーズはパーソナルコンピュータのひとつの形を定めたといえるでしょう．

このPCファミリーのうち，ホームコンピュータとして使えるように考えられた PC-6001 は，10万円台を割る低価格にもかかわらず，非常に高い機能が盛りこまれています．しかしその機能を十二分に引き出し，有効に使うかどうかは，ユーザー自身の態度いかんにかかっています．

PC-6001 を調べるにつれて，マニュアルに書かれていない有用な機能が多数あることに気付きました．また，PC-8001 に比べて機能が多少割愛された部分もあります．有用な機能を引き出し，割愛された機能を補うためには，どうしても N_{60}-BASIC の内部を知る必要があります．また，機械語を使用する場合においても内部構造を熟知していなければなりません．

本書は，こうした点に主眼をおいて PC-6001 を徹底的に活用するためのテクニカル・ノウハウをまとめたものです．読者の方々にパーソナルコンピュータに対する理解をより深めていただき，自由自在に使いこなすための書として利用していただければ幸いに存じます．

なお，本書を執筆するに際して，第 5 章を樋口理，N_{60}-BASIC の解析および付録を八木良一，他を一零が担当しました．さらに，システムソフト・スタッフが編集・監修を行ないました．

末筆ながら，著者らが本書の出版にあたり，大変御世話になりました株式会社アスキー出版のスタッフの方々ならびに株式会社システムソフト福岡の樺島社長，藤田出版部長に心から感謝いたします．

1982年 7 月

著者代表　一(かずと)　零(れい)

目　次

目　次

目　次

第１章　ＰＣ－6001のハードウェア仕様

１－１　本体ブロック図

１－２　システム仕様

１－３　プログラムエリア

１－４　ＲＯＭ ＆ ＲＡＭカートリッジ

１－５　I／Oマップ

第1章　PC-6001のハードウエア仕様

1-1　本体ブロック図

本体のブロック図を次のページに示します．

1-2　システム仕様

（1）　CPU

μPD780C-1(Z-80 コンパチブル)3.9936MHz

（2）　メモリ

○ROM……N_{60}-BASIC インタプリタ　16KB(μPD2364×2　マスクNO. 677, 678)

　　　　オプション　16KB(ROM カートリッジ)

○RAM……16KB(μPD416-3×8)

　　　　オプション　16KB(ROM & RAM カートリッジ)

（3）　CRT

○VDG(ビデオディスプレイジェネレータ)……M5C6847P-1(MC6847 コンパチブル)

○キャラクタ・ROM……4KB(μPD2332×1 マスク NO. 414)

○スクリーン構成……32文字×16行（512文字）

○文字構成……文字＋グラフィック記号248種(7×9ドット)

○グラフィック……256×192ドット

　　　　　　　　128×192ドット

　　　　　　　　 64× 48ドット

※グラフィックと文字の混在可能

○カラー……最大9色(黒，緑，黄，青，赤，白，シアン，マゼンダ，オレンジ)

○ページ数……最大4ページ

（4）　サウンド機能

○AY-3-8910 使用(クロック　3.579545MHz)

○音階……8オクターブ(3重和音出力可能)

○特殊効果音は8910の機能に準ず

（5）　カセットインタフェイス

○FSK 方式

○600ボー，1200ボー切り換え可

○リモート機能有

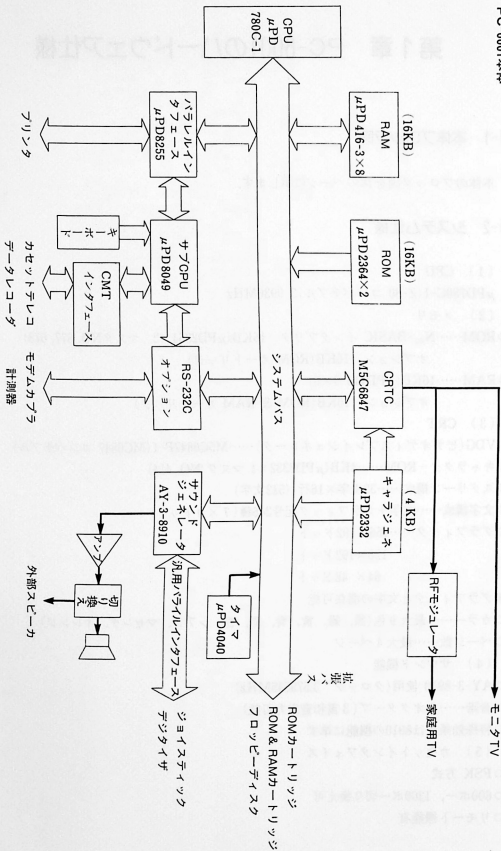



図1-1　PC-6001本体ブロック図

（6） プリンタ・インタフェイス

○TTLレベル8ビットパラレル(セントロニクス社仕様準拠)

○JIS コード準拠

（7） 汎用バス

○TTL レベルパラレルポート

ジョイスティック，デジタイザ等接続可

（8） CRTインタフェイス

○コンポジットビデオ信号出力方式(専用ディスプレイ)

○NTSC 出力方式(家庭用テレビ 1ch，または 2ch 使用)

（9） タイマ機能

○約 2ms ごとのインターバルタイマ(禁止可)

(10) RS-232C インタフェイス(オプション)

○EIA RS-232C 準拠

300，600，1200，2400，4800ボー選択可

(11) キーボード

○μPD8049 によるソフトウェアスキャン

○71キー JIS 標準配列準拠

○コントロールキー，ファンクションキー有

○リピート機能有

(12) 拡張バス

○2.54mm ピッチ 50 ピンバス

1-3 プログラムエリア

（1） 16K バイトの場合　　　　32K バイトの場合

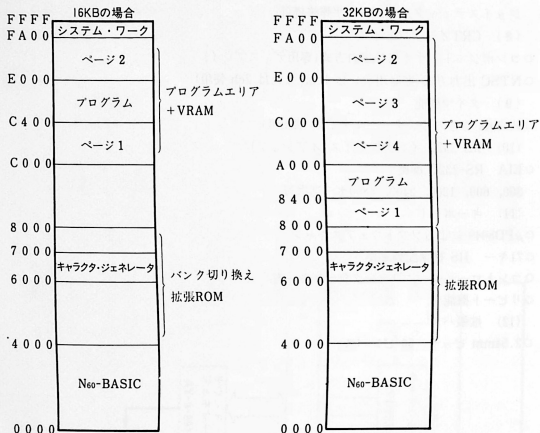



図1-2　メモリ・マップ

（2） プログラムエリアは使用するページ数で変わります.

ページ数 / RAM	1	2	3	4
16KB	C400～F9FFH	C400～DFFFH		
32KB	8400～F9FFH	8400～DFFFH	8400～BFFFH	8400～9FFFH

図1-3　プログラムエリアとページ数

1-4 ROM & RAM カートリッジ

（1） ユーザーエリア

PC-6006(ROM & RAM カートリッジ)を使用することによりRAMを16Kバイト増やすことができます．また，ROMを最大32Kバイトまで実装することが可能です．

RAM ＼ ページ数	32KB(増設有)	16KB(増設なし)
1	30140バイト	13756バイト
2	23484 〃	7100 〃
3	15292 〃	
4	7100 〃	

図1-4 PC-6006とユーザーエリアとの関係

（2） ROMの使用

ROMはマスクROMとしてμPD2316，μPD2332，μPD2364が使用できます．

われわれユーザーが利用する場合は，PROMを使用することになります．PROMとして使用できるのは，ROM & RAMカートリッジのマニュアルによればμPD2732となっていますが，μPD2716，TMS2532，TMS2564 等も使用可能です．

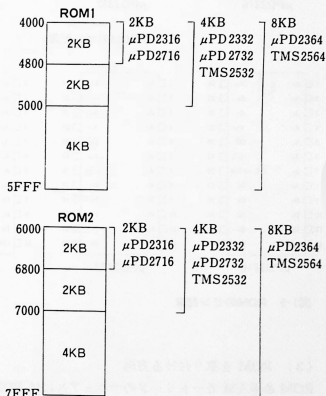



図1-5-1 拡張ROMメモリ・マップ

ROMサイズ	ROM名	スイッチの位置
2 KB	μPD2316	下
	μPD2716	下
4 KB	μPD2332	下
	μPD2732	上
	TMS2532	下
8 KB	μPD2364	下
	TMS2564	下

図1-5-2 ROMセレクトスイッチの位置

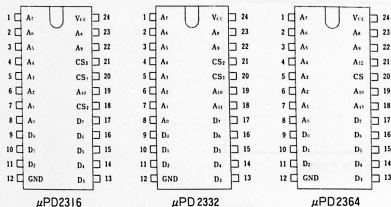
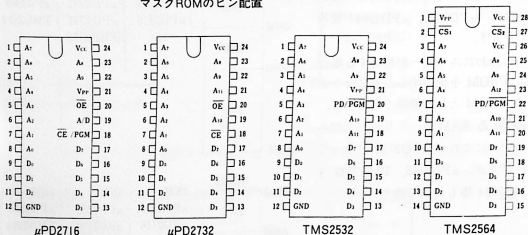



図1-6 ROMのピン配置

（3） ROM を取り付ける方向

ROM & RAM カートリッジのマニュアルには ROM を取り付ける方向の説明がありません．ROM の方向を間違えますと ROM が破損する原因になります．方向を間違えないように，また，ROM の足を曲げたりしないように注意深く取り付ける必要があります．

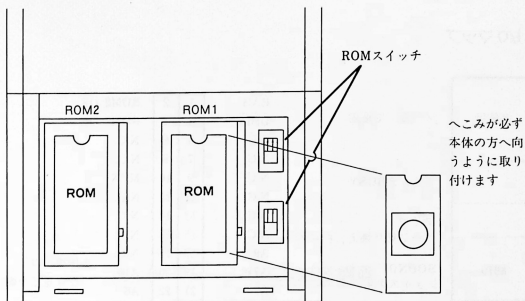



図1-7　ROMの取り付け方向

※PROM の種類によってはジャンパ線等を取り付ける必要があります.

1-5 I/Oマップ

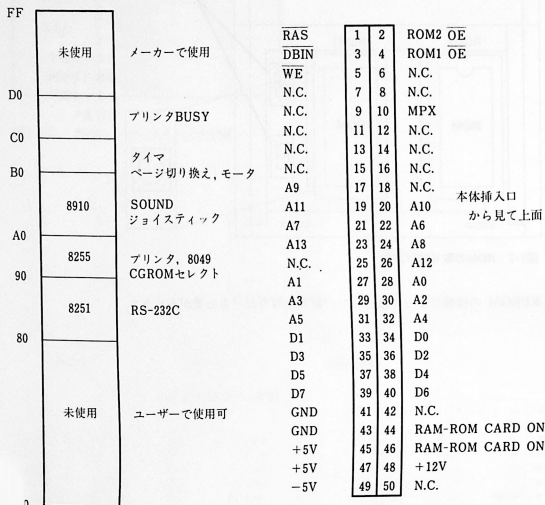



図1-8 I/Oマップ　　　図1-8-1 拡張バス信号

80～CFH までを PC-6001 の内部で使用しています．D0～FFH までは，ディスク等の周辺機器に使用すると思われます．00～7FH まではユーザー開放されたエリアで，それを使用するための外部バスの信号は以上のようになっています．

※I/O の機能については各章で説明が加えられています．

第２章　N_{60}－BASICの内部構造

第2章　N60-BASICの内部構造

2-1　メモリ・マップ

PC-6001内部のメモリ空間は，64Kバイトあり，前半の32KバイトがROM，後半32KバイトがRAMに割り当てられており，6000～6FFFHをバンク切り換えでキャラジェネのエリアとして使用しています．メモリ・マップはRAMの拡張やスクリーンのページ数によって変わりますが，とくにことわりがない場合を除いてRAM32Kバイト，ページ2で説明します．

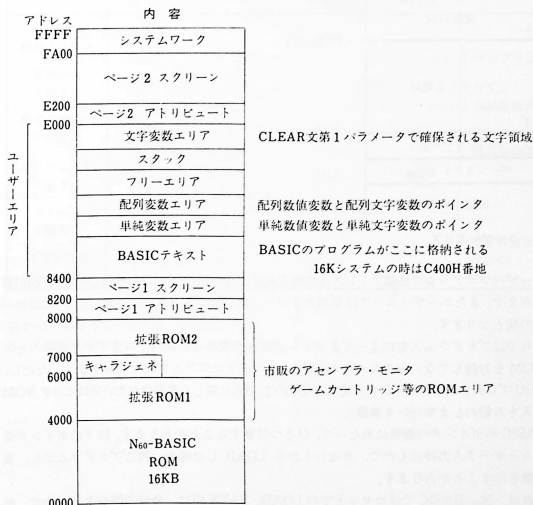



図2-1　メモリ・マップ

2-2　ユーザーエリア

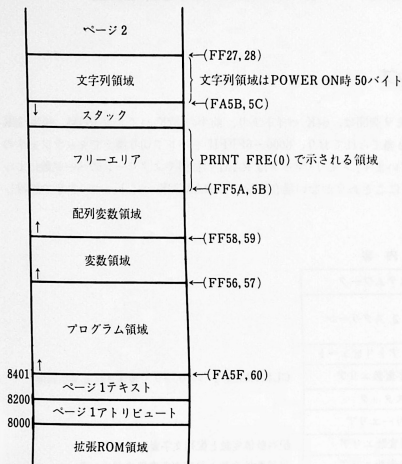



図2-2　ユーザーエリア

ユーザープログラムの先頭アドレスは拡張RAMがない場合, C401H, 拡張した場合, 8401Hになります. またユーザーエリアは使用するページ数で異なりますが, 7100バイト～30140バイトの間となります.

それではプログラム入力によって各ポインタがどう変わるか調べてみますが, 市販のモニタROMをお持ちでない方のためにBASIC・モニタプログラムを載せておきます. ただし, BASICプログラムは各ポインタを変化させるので, さらに詳しく学習されたい方はモニタROMの購入をお勧めします(付-8参照).

BASICのポインタの説明にあたって, ひとつ注意することがあります. 以下のポインタ値は, キーボード入力時のもので, カセットからLOAD した場合, 同じプログラムでも, 異なる値を示すことがあります.

これは, N_{60}-BASICではカセットでのLOAD, SAVE時に, 余分の00が入るためで, 詳しくは第6章で述べることにします.

```
100 REM カケザン
110 DIM C(12)
120 INPUT "a= ";A
130 INPUT "b= ";B:C(0)=A*B
140 PRINT "axb=";C(0)
150 END
Ok
RUN
a= ? 256
b= ? 16
axb= 4096
Ok
```

上記のサンプルプログラムで，その入力前，プログラム入力後，実行後のポインタの変化を調べてみましょう．

	ポインタアドレス	電源ON時 (RESET)	プログラム入力後	プログラム実行後	備考
プログラム開始アドレス	FA5F,60	8401			］プログラム領域
変数領域の始まり	FF56,57	8403	8458		］単純変数領域
配列変数領域の始まり	FF58,59	8403	8458	8466	］配列変数領域
フリーエリアの始まり	FF5A,5B	8403	8458	84AE	
スタックの始まり	FA5B,5C	DFCD			］文字列領域
文字列領域の始まり	FF27,28	DFFF			(50バイト)

図2-3　各ポインタの変化
注：RAM32KBでページ2を指定した時の値です

2-3 プログラムの格納状態

次のサンプルプログラムがメモリ内にどのように格納されているか説明します.

```
8400 00 0D 84 64 00 8E 20 B6 B9 BB DE DD 00 19 84 6E
8410 00 85 20 43 28 31 32 29 00 27 84 78 00 84 20 22
8420 61 3D 20 22 3B 41 00 3E 84 82 00 84 20 22 62 3D
8430 20 22 3B 42 3A 43 28 30 29 D2 41 CC 42 00 50 84
8440 8C 00 95 20 22 61 78 62 3D 22 3B 43 28 30 29 00
8450 56 84 96 00 80 00 00 00 41 00 00 00 00 00 89 42
8460 00 00 00 00 00 85 43 00 44 00 01 0D 00 00 00 00
```

	アドレス	データ	内容	
00行	8400		プログラム先頭	100 REM カケザン
	8401	0D	840D… リンクポインタ	(次の行の開始アドレス)
	8402	84		
	8403	64	0064… 行番号	(10進数に変換すると100)
	8404	00		
	8405	8E	REMの中間言語	
	8406	20	スペース	
	8407	B6	"カ"	
	8408	B9	"ケ"	
	8409	BB	"サ"	
	840A	DE	"〃"	
	840B	DD	"ン"	
	840C	00	行の終わりを示す	
110行	840D	19	8419… リンクポインタ	110 DIM C(12)
	840E	84		
	840F	6E	006E… 行番号	
	8410	00		
	8411	85	DIMの中間言語	
	8412	20	スペース	
	8413	43	"C" 変数名	
	8414	28	"(" 左カッコ	
	8415	31	"1" 配列の数(ASCIIコード)	
	8416	32	"2"	
	8417	29	")" 右カッコ	
	8418	00	行の終わり	

図2-4 プログラムの格納状態

リンクポインタが0000Hになったアドレスがプログラムテキストの終了アドレスと解釈され，その次のアドレスが変数領域の始まりとなります．

プログラムの格納のされ方は，PC-8001 の N-BASIC とだいたい同じですが，細部で異なる部分があります．

(1) GOTO, GOSUB, THEN の後に続く行番号

(2) 数値

この2つが，N-BASIC と比べて大幅に変わっています．

(1)の飛び先の行番号は N_{60}-BASIC では，行番号が ASCII コードで入っています．そして，その ASCII コードで入っている行番号を2バイトの16進数に変換し，現在の行番号より小さければプログラムの先頭から，行番号が大きければ，現在の行の後から，一致する行番号を探しますので，飛び先の行番号によって処理する時間が変わります．

```
LIST

10 GOTO 100
20 END
100 PRINT "100"
110 END
Ok
```

それでは上記のプログラムで，実行前と実行後での行番号の変化を調べてみましょう．

N-BASICの場合

N_{60}-BASICの場合

このようにN_{60}-BASICでは飛び先を毎回1行ごとに調べていきますのでGOTO, GOSUBを多用したプログラムは実行速度が遅くなります.

また，数値の格納のされ方は，N-BASICとN_{60}-BASICではまったく異質のものとなっています．これはN-BASICが数値として整数，単精度，倍精度が使用できるために，その型に合わせてメモリに格納されることに起因しています.

例としてAに1234567890を代入したときの N-BASIC と N_{60}-BASIC との違いを調べてみます.

N_{60}-BASICの場合

```
10 a=1234567890:print a
LIST
10 A=1234567890:PRINT A
Ok
RUN
 1.23456789E+09
Ok
```

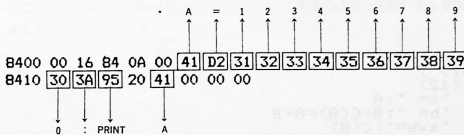
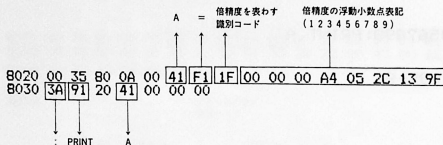
N-BASICの場合

```
10 a=1234567890:print a
list
10 A=1234567890#:PRINT A
Ok
run
1.23457E+09
Ok
```

N_{60}-BASICでは，入力した値とLISTしたときの値は同じになっています．メモリ内部も入力した値がそのまま入っていて，変数Aに数値を代入するときに5バイトの浮動小数点表記に変換されます．このため例のように，代入する数値が10桁以上の場合は，指数表示になります．

N-BASICの場合6桁以上だと，とくに指定していない限り倍精度扱いになります．そのためLISTしたときに倍精度数値を表わす〝#〟が数値の後ろに付き，代入する変数Aが単精度のときは，倍精度から単精度へ変換され6桁の指数表示となります．

これらの変化をメモリ上で調べてみましょう．

N_{60}-BASICの場合

N-BASICの場合

2-4 プログラムの回復のさせ方

BASICのプログラムを誤って消すことがあります．この原因としては

1.電源スイッチを切った．電源プラグが抜けた．停電した．

2.NEWを実行した．

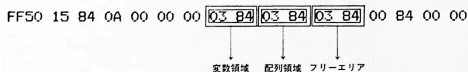
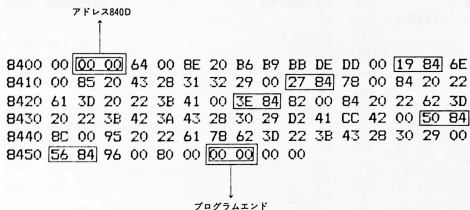
3.RESETボタンを押した．

等が考えられますが，1については絶対に元には戻りません．2については回復可能です．問題は3の場合です．プログラムが7100バイト以下ならば回復可能ですが，もし7100バイト以上のときは回復しません．これはRESETをかけるとイニシャライズ(初期設定)プログラムが働くためです．このときにページ1～4までをモード1の状態にセットします．それによって8000～83FFH，A000～A3FFH，C000～C3FFH，E000～E3FFHのVRAMの内容が書き換えられます．もし，この部分にBASICのプログラムがあった場合，書き換えられてしまいます．このようになってあわてるよりも　頻繁にCSAVEしておくことをお勧めします．

では，2の場合の回復の仕方について説明しましょう．

先ほどのプログラムをNEWしてみます．

```
LIST

100 REM カケザン
110 DIM C(12)
120 INPUT "a= ";A
130 INPUT "b= ";B:C(0)=A*B
140 PRINT "axb=";C(0)
150 END
Ok
new
OK

■
```

メモリ格納状態

※□で囲んだ部分が初期値にもどっています

メモリダンプから分かるように最初のリンクポインタの値が0000Hになっており，変数領域等のポインタの値が初期値に戻っています．これらのポインタ値を戻せばプログラムが復活するわけです．

ここでこれらのポインタの戻し方について説明します．

1.8405H 番地(16KBのときはC405H)から1バイトずつ読み出して00を書き込んであるアドレスを探し出す．

2.そのアドレス＋1の値を下位，上位の順に 8401,02H 番地に格納する(これがリンクポインタです)．

3.この時のリンクポインタを使ってエンドマーク(0000H)を見つける．

4.見つかったアドレス＋2の値を下位，上位の順に FF56,57H 番地にセットする．

以上の処理を行なうことによってプログラムが回復します．

この一連の処理を BASIC でプログラミングするのは無意味なので，機械語で行なってみました．

```
アドレス マシン語                 ニーモニック
0000 210484          LD   HL,8404H
0003 23       LOOP1: INC  HL
0004 7E              LD   A,(HL)
0005 B7              OR   A
0006 20FB            JR   NZ,LOOP1
0008 23              INC  HL
0009 220184          LD   (8401H),HL
000C 23       LOOP2: INC  HL
000D 23              INC  HL
000E 2256FF          LD   (0FF56H),HL
0011 2B              DEC  HL
0012 7E              LD   A,(HL)
0013 B7              OR   A
0014 C8              RET  Z
0015 2B              DEC  HL
0016 6E              LD   L,(HL)
0017 67              LD   H,A
0018 18F2            JR   LOOP2
```

このプログラムはリロケータブルになっていますので任意の番地で使用できます．

例　DE00H 番地から入力する．

例　DE00 番地のとき

```
DE00 21 04 84 23 7E B7 20 FB 23 22 01 84 23 23 22 56
DE10 FF 2B 7E B7 C8 2B 6E 67 18 F2 FF FF FF FF FF FF
```

```
EXEC&Hde00
Ok
LIST
100 REM カケザン
110 DIM C(12)
120 INPUT "a= ";A
130 INPUT "b= ";B:C(0)=A*B
140 PRINT "axb=";C(0)
150 END
Ok
```

2-5 中間言語

N_{60}-BASICでは命令(キーワード)は，メモリ節約，処理速度向上のため，中間言語と呼ばれる1バイトのコードで格納されます.

```
100 INPUT A
110 C=SQR(A)
120 PRINT C

8400 00 09 84 64 00 84 20 41 00 14 84 6E 00 43 D2 DC
8410 28 41 29 00 1C 84 78 00 95 20 43 00 00 00 00 00
```

※___で示される部分が中間言語です

命令(KEYWORD)	中間言語
INPUT	84
=	D2
SQR	DC
PRINT	95

図2-5 キーワードと中間言語の対応

キーワードは 021D～036DH 番地までデータとして ASCII コードで入っています.

次のデータ

	2文字目以降		

最初の文字は最上位ビットが1になっている

図2-6 データの形式

それでは実際に格納されているメモリ内容を調べてみましょう.

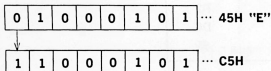
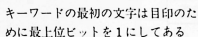
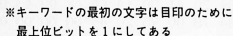



図2-7　キーワードの格納状態

N₆₀-BASIC の命令(キーワード)を出力するプログラムを次に紹介します．

```
10 FOR I=&H21D TO &H36D
20 A=PEEK(I):IF A>128 THEN A=A AND &H7F:PRINT
30 PRINT CHR$(A);
40 NEXT I
```

次に，キーワードと中間言語の対応を調べるプログラムを示します．

```
 10 AD=&H21D:FOR I=&H80 TO &HAA
 20 DA=I:GOSUB 200:PRINT HE$;TAB(5);
 30 A=PEEK(AD):PRINT CHR$(A AND &H7F);:AD=AD+1
 40 A=PEEK(AD):IF A>127 THEN 60
 50 PRINT CHR$(A);:AD=AD+1:GOTO 40
 60 PRINT TAB(14);CHR$(I):NEXT I
100 FOR I=&HC2 TO &HF1
110 DA=I:GOSUB 200:PRINT HE$;TAB(5);
120 A=PEEK(AD):PRINT CHR$(A AND &H7F);:AD=AD+1
130 A=PEEK(AD):IF A>127 THEN 150
140 PRINT CHR$(A);:AD=AD+1:GOTO 130
150 PRINT TAB(14);CHR$(I):NEXT I
160 END
```

```
200 REM hex$
210 HE$="00":HE=INT(DA/16):HF=HE:GOSUB 230
220 HE=DA-(16*HF):GOSUB 230:HE$=RIGHT$(HE$,2):RE
    TURN
230 IF HE>9 THEN HE=HE+55:GOTO 250
240 HE=HE+&H30
250 HE$=HE$+CHR$(HE):RETURN
```

```
80   END       ♠     CA   +        ハ
81   FOR       ♥     CB   -        ヒ
82   NEXT      ♣     CC   *        フ
83   DATA      ♦     CD   /        ヘ
84   INPUT     ○     CE   ^        ホ
85   DIM       ●     CF   AND      マ
86   READ      を    D0   OR       ミ
87   LET       ぁ    D1   >        ム
88   GOTO      ぃ    D2   =        メ
89   RUN       ぅ    D3   <        モ
8A   IF        ぇ    D4   SGN      ヤ
8B   RESTORE   ぉ    D5   INT      ユ
8C   GOSUB     ゃ    D6   ABS      ヨ
8D   RETURN    ゅ    D7   USR      ラ
8E   REM       ょ    D8   FRE      リ
8F   STOP      っ    D9   INP      ル
90   OUT             DA   LPOS     レ
91   ON        あ    DB   POS      ロ
92   LPRINT    い    DC   SQR      ワ
93   DEF       う    DD   RND      ン
94   POKE      え    DE   LOG      ゛
95   PRINT     お    DF   EXP      ゜
96   CONT      か    E0   COS      た
97   LIST      き    E1   SIN      ち
98   LLIST     く    E2   TAN      つ
99   CLEAR     け    E3   PEEK     て
9A   COLOR     こ    E4   LEN      と
9B   PSET      さ    E5   HEX$     な
9C   PRESET    し    E6   STR$     に
9D   LINE      す    E7   VAL      ぬ
9E   PAINT     せ    E8   ASC      ね
9F   SCREEN    そ    E9   CHR$     の
A0   CLS             EA   LEFT$    は
A1   LOCATE    。    EB   RIGHT$   ひ
A2   CONSOLE   「    EC   MID$     ふ
A3   CLOAD     」    ED   POINT    へ
A4   CSAVE     、    EE   CSRLIN   ほ
A5   EXEC      ・    EF   STICK    ま
A6   SOUND     ヲ    F0   STRIG    み
A7   PLAY      ァ    F1   TIME     む
A8   KEY       ィ
A9   LCOPY     ゥ
AA   NEW       ェ
C2   TAB(      ツ
C3   TO        テ
C4   FN        ト
C5   SPC(      ナ
C6   INKEY$    ニ
C7   THEN      ヌ
C8   NOT       ネ
C9   STEP      ノ
```

下＼上	8	9	A	B	C	D	E	F
0	♠ END	OUT	CLS	ー	タ	ミ OR	た COS	み STRIG
1	♥ FOR	あ ON	。 LOCATE	ア	チ	ム >	ち SIN	む TIME
2	♣ NEXT	い LPRINT	「 CONSOLE	イ	ツ TAB(	メ =	つ TAN	め
3	♦ DATA	う DEF	」 CLOAD	ウ	テ TO	モ <	て PEEK	も
4	○ INPUT	え POKE	、 CSAVE	エ	ト FN	ヤ SGN	と LEN	や
5	● DIM	お PRINT	・ EXEC	オ	ナ SPC(	ユ INT	な HEX$	ゆ
6	を READ	か CONT	ヲ SOUND	カ	ニ INKEY$	ヨ ABS	に STR$	よ
7	ぁ LET	き LIST	ァ PLAY	キ	ヌ THEN	ラ USR	ぬ VAL	ら
8	ぃ GOTO	く LLIST	ィ KEY	ク	ネ NOT	リ FRE	ね ASC	り
9	ぅ RUN	け CLEAR	ゥ LCOPY	ケ	ノ STEP	ル INP	の CHR$	る
A	ぇ IF	こ COLOR	ェ NEW	コ	ハ +	レ LPOS	は LEFT$	れ
B	ぉ RESTORE	さ PSET	ォ	サ	ヒ −	ロ POS	ひ RIGHT$	ろ
C	ゃ GOSUB	し PRESET	ャ	シ	フ *	ワ SQR	ふ MID$	わ
D	ゅ RETURN	す LINE	ュ	ス	ヘ /	ン RND	へ POINT	ん
E	ょ REM	せ PAINT	ョ	セ	ホ ∧	゛ LOG	ほ CSRLIN	
F	っ STOP	そ SCREEN	ッ	ソ	マ AND	゜ EXP	ま STICK	

図2-9　中間言語表(コード順)

2-6 中間言語処理ルーチン

BASIC プログラムを RUN させると N_{60}-BASIC インタプリタは中間言語に対応したジャンプテーブルを参照して処理先のアドレスを求めます．このジャンプテーブルはワークエリア上にあります．

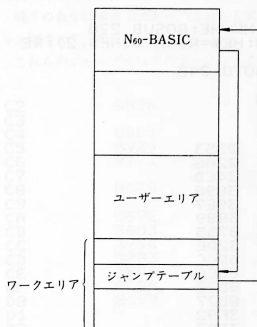



図2-10 ジャンプテーブルの位置

このジャンプテーブルは各2バイトで未使用分を含めて，110個(220バイト)あります．これらのジャンプテーブルは2つのブロックに分かれており，第1ブロックがコマンド・ステートメント用，第2ブロックが関数用になっています．

第1ブロックが，66個(132バイト)で FA61～FAE4H 番地を使用しており，中間言語の 80H～C1H に対応しています．第2ブロックは関数専用で44個(88バイト)あり，FAE5～FB3CH 番地を使用しており，D4～FFH までの中間言語が割り付けられています．

AB～C1H，F2～FFH までの中間言語に対応する命令は N_{60}-BASIC にはありませんが，将来，拡張 BASIC 等で使用されると思われます．

この，ジャンプテーブルを使用して N_{60}-BASIC の命令追加，機能強化をすることができます．

さて，これらのジャンプテーブルの値は 0195～021CH 番地にあり，イニシャライズの時に RAM 上に転送されます．

中間言語コードに対応する処理アドレスを求めるプログラムは次のとおりです．

```
 10 AD=&HFA60:FOR I=&H80 TO &HAA
 20 DA=I:GOSUB 200:PRINT HE$;TAB(10);:AD=AD+2
 30 DA=PEEK(AD):GOSUB 200:PRINT HE$;:AD=AD-1
 40 DA=PEEK(AD):GOSUB 200:PRINT HE$:AD=AD+1
 50 NEXT I
100 AD=&HFAE4:FOR I=&HD4 TO &HEC
110 DA=I:GOSUB 200:PRINT HE$;TAB(10);:AD=AD+2
120 DA=PEEK(AD):GOSUB 200:PRINT HE$;:AD=AD-1
130 DA=PEEK(AD):GOSUB 200:PRINT HE$:AD=AD+1
140 NEXT I
150 END
199 REM hex$
200 HE$="00":HE=INT(DA/16):HF=HE:GOSUB 220
210 HE=DA-(16*HF):GOSUB 220:HE$=RIGHT$(HE$,2):RE
    TURN
220 IF HE>9 THEN HE=HE+55:GOTO 240
230 HE=HE+&H30
240 HE$=HE$+CHR$(HE):RETURN
```

```
80        3535     A8        2353
81        067E     A9        22A6
82        35F7     AA        34CD
83        07E0     D4        3898
84        09AB     D5        39E7
85        3302     D6        38B9
86        0A09     D7        0755
87        07F5     D8        32DE
88        07A0     D9        0DCC
89        0781     D9        0DCC
8A        0861     DA        0D22
8B        3519     DB        0D27
8C        078F     DC        3F92
8D        07BC     DD        3BA3
8E        07E2     DE        3EA5
8F        3533     DF        3E21
90        0DD6     E0        3F51
91        0844     E1        3F57
92        087A     E2        3FD3
93        0D3A     E3        0DF3
94        0DFA     E4        3229
95        087E     E5        03EA
96        356B     E6        305B
97        05DB     E7        32BA
98        05D6     E8        3238
99        35A9     E9        3249
9A        1D9B     EA        3257
9B        2D3C     EB        3286
9C        2D37     EC        328F
9D        2DC7
9E        2EDC
9F        1E04
A0        1DF8
A1        1CD2
A2        1CF6
A3        2496
A4        247E
A5        261D
A6        1E9B
A7        1EB3
```

対応表を見て分かるように，中間言語でC2～D3H，ED～F1Hの処理アドレスがありません．これらはインタプリタ内部で特殊処理用に使われています．

NOT，FN，INKEY$ および POINT～TIME(ED～F1H)はコンペア(比較)命令によってインタプリタ内でチェックされており，個別にジャンプします．

また，TAB，TO，SPC，THEN，STEPは，この命令のみで使われることはありません．たとえば，SPC，TAB は，必ず PRINT 文と一緒に使われるので，PRINT の処理ルーチンの中で，TAB と SPC の処理を行なっています．

残りの命令は演算子であり，これらも式の中で使われますので，式解析ルーチンの中でジャンプさせています．

これらのジャンプ先は次のようになっています．

```
C2          0926
C3
C4          0D61
C5          0926
C6          2771
C7
C8          0CF9
C9
CA          367E
CB          3683
CC          37B4
CD          3803
CE          3EFA
CF          0C9A
D0          0C99
D1
D2
D3
ED          2D55
EE          0D30
EF          2236
F0          2256
F1          1E83
```

2-7 識別コード

N-BASICにおいては，数値の型を区別するために0B~1FHまで識別コードとして使っていますが，単精度や倍精度がないN_{60}-BASICでは，数値を区別するための識別コードがなくなっています．N_{60}-BASICで識別コードとして使用しているのは14Hで，これはキャラクタコードの00~1FHのグラフィックキャラクタのために使われています．この部分は本来，コントロールコードになっており，グラフィックキャラクタとコントロールコードとの区別をつけるために，識別子が用いられています．

実際にグラフィックキャラクタがどのように入っているか調べてみましょう．

```
10 PRINT "月火水木金土日年円時"

8400 00 1E 84 0A 00 95 20 22 [14 31][14 32][14 33][14 34]
8410 [14 35][14 36][14 37][14 38][14 39][14 3A] 22 00 00 00
```

"月"は14H+31Hの2バイトで入っています．これで分かるように14Hに続くキャラクタコードには30Hのボリュウムを持たせています．

このため"月"を画面にCHR$で表示するとき

PRINT CHR$(1)

では表示されずに

PRINT CHR$(&H14) ; CHR$(&H31)

で表示されます．

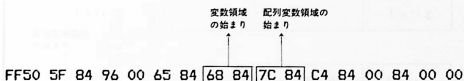
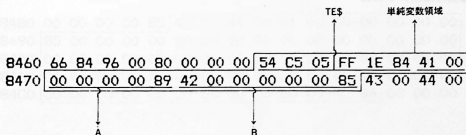
2-8　単純変数領域

単純変数領域は，プログラム領域の後ろより，始まります．プログラムを実行して実際にその変数が使われますと，この変数領域に登録されます．この領域は，FF56, 57H で示されるアドレスから，(FF58, 59H で示されるアドレス)－1までになります．

実際に単純変数の状態を調べましょう．

```
100 REM カケザン
110 DIM C(12):TE$="カケザン"
120 INPUT "A= ";A
130 INPUT "B= ";B:C(0)=A*B
140 PRINT TE$,"A×B=";C(0)
150 END
Ok
RUN
A= ? 256
B= ? 16
カケザン          A×B= 4096
Ok
```

変数領域の始まり　　配列変数領域の始まり

```
FF50 5F 84 96 00 65 84 [68 84] [7C 84] C4 84 00 84 00 00
```

このプログラムでは単純変数領域は 8468～847BH 番地までになっており，その格納のされ方は次のようになります．

このプログラムでは単純変数としては TE$，A，Bの３つがありました．それぞれメモリ内では，

```
                 変数名   変数の文字数  文字列の格納されているポインタ
TE$     8468     54 C5   05   FF  1E 84   (841EHより文字列があることを示している)
                 "T"  "E"
                        ↓
                      bit7を1にしてある
A       846E     41 00   00 00 00 00 89
                 変数名    数値256を表わす(浮動小数点表記)
                 "A"
B       8475     42 00   00 00 00 00 85   変数Aと同じ
```

のように格納されています.

N₆₀-BASICでは変数の種類が数値と文字との2つしかありませんので,文字型変数の変数名の2文字目をその区別に使用しており,最上位ビット(bit7)を1にすることによって文字型であることを示しています.

	変数名	数値またはディスクリプタ部分			バイト数
数値変数	2バイト	5バイト			7バイト
文字変数	2バイト	文字数	ダミー	ポインタ※	6バイト
		ストリングディスクリプタ			

※ポインタ文字列の格納先アドレスを示す

図2-11 単純変数の格納のされ方

2-9　配列変数領域

FF58, 59H で示されるアドレスから，(FF5A, 5BH で示されるアドレス)−1 が，配列変数領域になります．この領域は DIM 文を実行した場合，および配列添字が10以下の配列変数を使用したときに，その配列に見合ったメモリが確保されます．

当然，単純変数領域が増えると，それに合わせて配列変数領域は後ろに移動します．また，プログラムを変更すると領域はクリアされます．

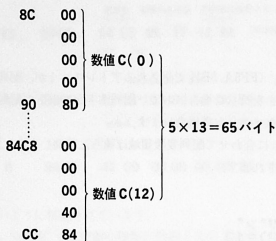
```
100 REM ｶｹｻﾞﾝ
110 DIM C(12):TE$="ｶｹｻﾞﾝ"
120 INPUT "A= ";A:C(12)=12
130 INPUT "B= ";B:C(0)=A*B
140 PRINT TE$,"A×B=";C(0)
150 END
Ok
RUN
A= ? 256
B= ? 16
ｶｹｻﾞﾝ           A×B= 4096
Ok
```

配列変数領域の始まり　　フリーエリアの始まり

```
FF58 [85 84][CD 84] 00 84 00 00
```

配列変数領域

```
8480 00 00 00 00 85 43 00 44 00 01 0D 00 00 00 00 00
8490 8D 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
84A0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
84B0 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00 00
84C0 00 00 00 00 00 00 00 00 00 00 00 40 84 00 00 00
```

8485	43	配列変数名"C"
86	00	
87	44	以降に使用するメモリ数(5×13＋3＝68→44H)
88	00	
89	01	…配列の次元数(ここでは1次元を表わす)
8A	0D	配列要素の大きさ　13…(添字の数＋1)
8B	00	

次に，配列の次元数が多次元のときについて調べてみましょう．

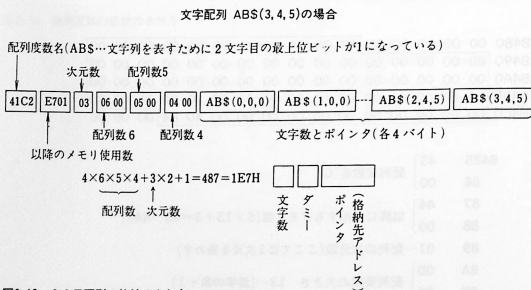
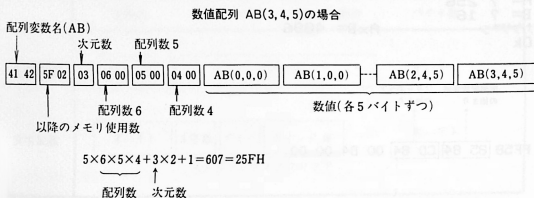



図2-12 多次元配列の格納のされ方

2-10　文字変数と文字列領域

変数が数値の場合は，その値にかかわらず５バイトの浮動小数点表記で表わされ，メモリ数に変化がありませんが，文字列のときは，０～255バイトまでの文字データが許されるために，文字列によってメモリ数が変化します．このため N_{60}-BASICでは，変数が文字列の場合，ストリングディスクリプタと呼ばれる４バイトのデータが変数領域に格納されます．この格納のされ方は２-８，２-９で説明したとおりです．

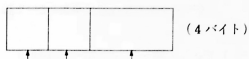



図２-13　ストリングディスクリプタの構成

文字列は，文字列領域に格納され，ポインタもこの領域を示しますが，文字列が，プログラム中にあるときに限って，ポインタはプログラムエリアを示します．このときは当然のこととして，文字列領域には格納されません．この処理があることがメモリの節約になっています．

```
100 A$="PC-6001"
110 FOR I=0 TO 2:READ NM$(I):NEX
T
120 DATA System,Soft,Fukuoka
130 INPUT "Phone No. ";PH$
Ok
RUN
Phone No. ? 092-714-6354
Ok
```

それでは，このプログラム実行後の変数のポインタを調べてみましょう．

まず文字列領域(CLEARの第１パラメータで指定)はFA5B, 5CHで示されるアドレス＋1からFF27,28Hで示されるアドレスまでです．

図2-14 文字列領域

スタックポインタの初期アドレス

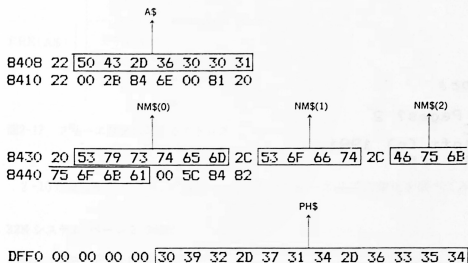
FF20 FF FF FF 00 00 01 00 FF DF 00 84 2D FF 07 FF B0

フリーエリア最終アドレス

DFCE~DFFFH までの50バイトが文字列領域になっているのが分かります．次に文字変数を調べてみます．

変 数 名	ストリングディスクリプタのアドレス	ストリングディスクリプタ		文字列格納場所
		文 字 数	ポインタ	
A$	8 4 6 0	7	8 4 0 9	プログラム領域
PH$	8 4 6 D	12	D F F 4	文字列領域
NM$(0)	8 4 7 8	6	8 4 3 1	プログラム領域
NM$(1)	8 4 7 C	4	8 4 3 8	〃
NM$(2)	8 4 8 0	7	8 4 3 D	〃

図 2-15　文字変数のディスクリプタ

格納場所のポインタを調べてみます.

PH$ は文字列領域の終わりの方から格納されていきます. 文字列データが変わった場合や, 新しい文字変数が使用されたときは, 前へと格納されていきます. そして文字列領域いっぱいになると, 不要となった文字列データをつめていきます. これをガベージコレクションと呼んでおり, このためにプログラムによってはしばらくの間, 実行が中断することがあります.

2-11 フリーエリア

フリーエリアを調べる命令として，FRE 関数があります．プログラムエリアの残りを調べるときは( )の内に数値を入れ，文字列領域の残りを知りたいときは文字変数を使います．

32KB ページ2のとき

```
How Many Pages? 2
N60-BASIC
By Microsoft (c) 1981
23484 Bytes free
Ok
?fre(0)
 23484
Ok
?fre(a$)
 50
Ok
```

16KB ページ2のとき

```
How Many Pages? 2
N60-BASIC
By Microsoft (c) 1981
7100 Bytes free
Ok
?fre(0)
 7100
Ok
?fre(a$)
 50
Ok
```

文字列領域はページ数，増設 RAM の有無に関係なく，初期値は50バイトになります．

N_{60}-BASIC 起動時のページ数とフリーエリアのバイト数の関係は次のようになります．

RAM	ページ数	フリーエリア	文字列領域Ⓐ	ユーザーメモリ上限Ⓑ
32KB	1	30140	50	F9FFH
	2	23484	50	DFFFH
	3	15292	50	BFFFH
	4	7100	50	9FFFH
16KB	1	13756	50	F9FFH
	2	7100	50	DFFFH

ⒶCLEARの第1パラメータで指定される

ⒷCLEARの第2パラメータで指定される

図2-16 ページ数とフリーエリア

これらフリーエリアに関するアドレスは，次のようになります．

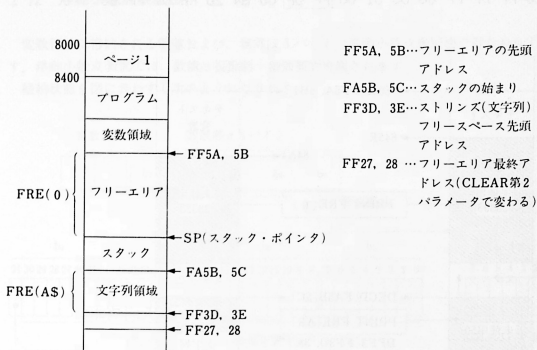



図2-17　フリーエリアに関するアドレス

2-10で使ったプログラムを実行して，そのフリーエリアの変化を調べてみましょう．

32K システム ページ2で起動

```
100 A$="PC-6001"
110 FOR I=0 TO 2:READ NM$(I):NEX
T
120 DATA System,Soft,Fukuoka
130 INPUT "Phone No. ";PH$
Ok
RUN
Phone No. ? 092-714-6254
Ok
PRINT fre(0),fre(a$)
 23323          38
Ok
```

プログラム実行後のポインタ

```
FF58 71 84 [A4 84] 44 84 00 00
FA58 01 0E 00 [CD DF] FF FF 01
```

```
FF38 FF 07 FF B0 40 [F3 DF] 27
FF20 FF FF FF 00 00 01 00 [FF DF] 00 84 2D FF 07 FF B0
```

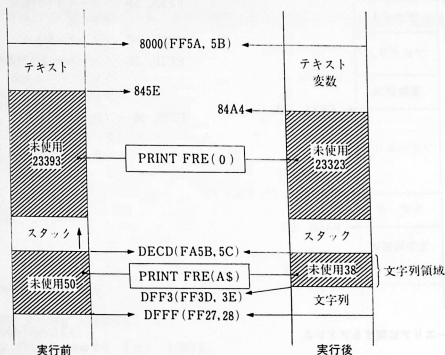



図2-18 フリーエリアの変化

実行後，フリーエリア(プログラムエリアの残り)が70バイト(23393−23323=70)，文字列領域が12バイト(50−38=12)減っています．フリーエリアの70バイトの内訳は，6＋7＋51＋6＝70になります．文字列領域の方は，PH$の"092-714-6254"の12バイトが文字列領域に書き込まれています．

2-12 浮動小数点表記法

変数領域に格納される数値および，演算は5バイトの浮動小数点表記法で行なわれています．浮動小数点表記では，数値は仮数部＋指数部で表現されます．

格納状態を図に表わすと次のようになります．

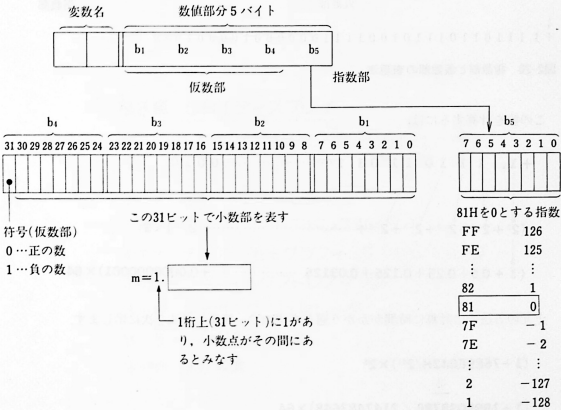



図2-19 浮動小数点の表記法

これにより数値nをもとめる式は，

$$n = \text{Im} \times 2^r$$

となります．

実際に数値を入れてみましょう．

```
cd=123.456789
Ok
```

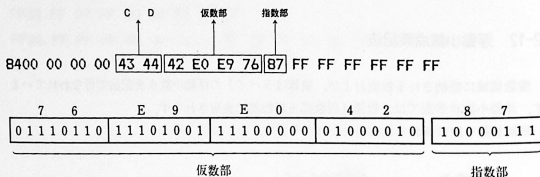
↓

$+1.111011011101001111000000001000010\times2^{6}$

図2-20 指数部と仮数部の表現法

この値を計算するには，

+1，1 1 1 0 1 1 0 1 1 ………………………0 0 1 0

$(2^{0}+2^{-1}+2^{-2}+2^{-3}+2^{-5}+ \cdots\cdots 2^{-30})\times2^{6}$

$(1+0.5+0.25+0.125+0.03125 \cdots\cdots +0.0000000001)\times64$

上記の方法では計算に時間がかかり過ぎますので，簡単な方法を次に示します．

$(1+76E9E042H/2^{31})\times2^{6}$

⇓

(1+1995038786／2147483648)×64

⇓

(1+0.92901233) ×64＝123.456789

となり代入値と等しいことが分かります．

第3章　CRTディスプレイ

第3章　CRTディスプレイ

3-1　VDG(ビデオディスプレイジェネレータ)

（1） CRT コントローラの IC として M5C6847P-1(MC6847 と同等品)が使用されています. この VDG は家庭用 TV に接続するのを目的としてつくられたディスプレイ用の IC です.

機能としては次のようなものがあります.

イ.32文字×16行(512文字)のアルファニューメリック表示.
ロ.グラフィック表示モード.
ハ.64文字のキャラクタ・ジェネレータ内蔵.

VDG は 4 種類のアルファニューメリック表示と 8 種のグラフィック表示の機能を持っており, 各モードの一覧を図 3 - 1 に示します.

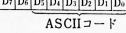
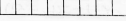
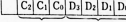
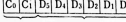
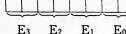
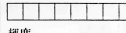
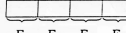
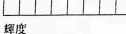
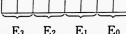
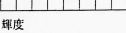
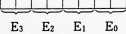
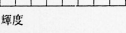
このモードの内, N_{60}-BASIC では外部アルファニューメリック, セミグラフィック 6 , 128×192カラーグラフィック, 256×192グラフィックモードをサポートしています.

表示(1画面256×192ドット)		データ	モード	N60-BASICのモード
モード	内容(数字はドット単位の長さ)			
32文字×16行	8, 5, 7, 12 5×7ドット1文字	D7 D6 D5 D4 D3 D2 D1 D0 ASCIIコード	内部アルファニューメリック	——
32文字×16行	8, 12 8×12ドット1文字	外部キャラジュネのアドレス	外部アルファニューメリック	1
64×32ドット	4, 6 D3 D2 / D1 D4 4画素は同色	C2 C1 C0 D3 D2 D1 D0 カラー 輝度	セミグラフィック4	——
64×48ドット	4, 4 D5 D4 / D3 D2 / D1 D0 6画素は同色	C0 C1 D5 D4 D3 D2 D1 D0 カラー 輝度	セミグラフィック6	2
64×64ドット	E3 E2 E1 E0 (4, 3)	E3 E2 E1 E0	64×64カラーグラフィック	——
128×64ドット	D7 D6 D5 D4 D3 D2 D1 D0 (2, 3)	輝度	128×64グラフィック	——
28×64ドット	E3 E2 E1 E0 (2, 3)	E3 E2 E1 E0	128×64カラーグラフィック	——
128×96ドット	D7 D6 D5 D4 D3 D2 D1 D0 (2, 2)	輝度	128×96グラフィック	——
128×96ドット	E3 E2 E1 E0 (2, 2)	E3 E2 E1 E0	128×96カラーグラフィック	——
128×192ドット	D7 D6 D5 D4 D3 D2 D1 D0 (2, 1)	輝度	128×192グラフィック	——
128×192ドット	E3 E2 E1 E0 (2, 1)	E3 E2 E1 E0	128×192カラーグラフィック	3
256×192ドット	D7 D6 D5 D4 D3 D2 D1 D0 (1, 1)	輝度	256×192グラフィック	4

図3-1　全モード一覧

3-2 アトリビュート

アトリビュート(属性)エリアに格納されているデータによって，カラーおよび表示モードを設定します．アトリビュートエリアは512(32×16)バイト使用しており，そのアドレスは，図3-2で示されるとおりです．

ページ	RAM16KB	RAM32KB
1	C000～C1FFH	8000～81FFH
2	E000～E1FFH	E000～E1FFH
3		C000～C1FFH
4		A000～A1FFH

図3-2　アトリビュートエリア

アトリビュートメモリの出力はVDGのコントロール信号に接続されており，このアトリビュートの値を変えることによってモードを変更します．

各コントロール信号は次のように接続されています．

7	6	5	4	3	2	1	0 データビット
$\overline{\text{A}}$／G	$\overline{\text{A}}$／S	$\overline{\text{INT}}$／ENT	GM0	GM1	GM2	CSS	INV

図3-3　コントロール信号の接続

INV　アルファニューメリック時，バックと文字の色を変える。

CSS　カラーセットセレクト入力(色相を180°変えて色を変える)。

GM0～GM2　グラフィックモード切り換え。

$\overline{\text{INT}}$／ENT
- キャラクタ・モード時CGを内部または外部の選択。
  - 0＝内部
  - 1＝外部
- セミグラフィック時セミグラフィック4モードか6モードの選択。
  - 0＝4モード
  - 1＝6モード

$\overline{\text{A}}$／S　アルファニューメリックかセミグラフィックの切り換え。
- 0＝アルファニューメリック
- 1＝セミグラフィック

$\overline{\text{A}}$／G　アルファニューメリックとグラフィックの切り換え。
- 0＝アルファニューメリック
- 1＝グラフィック

アトリビュートの初期値は次のようになります．

モード1…20H　　　モード3…8CH

モード2…60H　　　モード4…DCH

これを出力するプログラムを紹介します．

アトリビュートを調べるプログラム

```
 10 FOR I=1 TO 4:SCREEN I,2,2:CLS:A(I)=PEEK(&HE0
    00):NEXT I:SCREEN 1,1,1
 20 FOR I=1 TO 4:DA=A(I):GOSUB 200:PRINT "モード";
    I;"= ";HE$;"H":NEXT
 30 END
200 REM hex$
210 HE$="00":HE=INT(DA/16):HF=HE:GOSUB 230
220 HE=DA-(16*HF):GOSUB 230:HE$=RIGHT$(HE$,2):RE
    TURN
230 IF HE>9 THEN HE=HE+55:GOTO 250
240 HE=HE+&H30
250 HE$=HE$+CHR$(HE):RETURN
```

実行結果

```
RUN
モード  1= 20H
モード  2= 60H
モード  3= 8CH
モード  4= DCH
Ok
```

次に各コントロール記号を説明します．

（1） **INV(bit0)**

テキスト画面のみに働き，文字とバックの色を反転させます．

```
10 SCREEN 1,2,2:COLOR 1:CLS
20 FOR I=32 TO 255:PRINT CHR$(I);:NEXT I
30 FOR I=&HE000 TO &HE1FF:A=PEEK(I):POKE I,A OR
    1:NEXT I
```

上記のプログラムを実行すると，緑のバックに白の文字であったのが，白のバックに緑の文字に変化します(実際は白色はやや黄色を帯びて表示される場合もあります)．

これは"COLOR 2"を実行したのと同じ結果になります．

（2） **CSS(bit1)**

モード1～4まですべてに対して機能し，色相を180°変えることによって，色を変えることができます．これは COLOR 命令の3番目のパラメータに対応しています．

```
10 SCREEN 1,2,2:COLOR 1:CLS
20 FOR I=32 TO 255:PRINT CHR$(I);:NEXT I
30 FOR I=&HE000 TO &HE1FF:A=PEEK(I):POKE I,A OR
    2:NEXT I
```

上記のプログラムを実行すると分かるように，バックを緑からオレンジへ変化させます．

（3） **GM0～GM2(bit4～2)**

グラフィックのモードを切り換えます．グラフィックモードにするには $\bar{A}$/G 信号を"1"にする必要があります．

GM 0	GM 1	GM 2	グラフィックモード
0	0	0	64× 64カラーグラフィック
1	0	0	128× 64グラフィック
0	1	0	128× 64カラーグラフィック
1	1	0	128× 96グラフィック
0	0	1	128× 96カラーグラフィック
1	0	1	128×192グラフィック
0	1	1	128×192カラーグラフィック
1	1	1	256×192グラフィック

図3-4　グラフィックモードの切り換え

（4） **$\overline{\text{INT}}$/ENT(bit5)**

この信号は2つの使い方があり，キャラクタ・モード(テキストモード)では，VDG 内部の CG(キャラクタ・ジェネレータ)を使用するか，外部の CG を使用するかの選択に使います．また，セミグラフィックモードでは，4モードと6モードの切り換えに用いられています．

まず，キャラクタ・モードについて説明しましょう．

PC-6001 では外部 CG を使用していますので，このビットは"1"になりますが，"0"にすることによって内部 CG に切り換えることができます．

```
10 SCREEN 1,2,2:COLOR 1:CLS
20 FOR I=32 TO 255:PRINT CHR$(I);:NEXT I
30 FOR I=&HE000 TO &HE1FF:A=PEEK(I):POKE I,A AN
   D &HDF:NEXT I
```

このプログラムを実行すると，画面の文字が小さくなっていきます．ところがカナやグラ

フィックのキャラクタであった所が英文字や英記号を表示してしまいます．これは内部キャラクタが64文字(6bit ASCII)しかないために，ASCII コードに対応するイメージが発生してこのようになったわけです．

データ(H)	キャラクタ	データ(H)	キャラクタ	データ(H)	キャラクタ	データ(H)	キャラクタ
0 0	ⓐ	1 0	P	2 0	SP	3 0	0
0 1	A	1 1	Q	2 1	!	3 1	1
0 2	B	1 2	R	2 2	〃	3 2	2
0 3	C	1 3	S	2 3	#	3 3	3
0 4	D	1 4	T	2 4	$	3 4	4
0 5	E	1 5	U	2 5	%	3 5	5
0 6	F	1 6	V	2 6	&	3 6	6
0 7	G	1 7	W	2 7	,	3 7	7
0 8	H	1 8	X	2 8	(	3 8	8
0 9	I	1 9	Y	2 9	)	3 9	9
0 A	J	1 A	Z	2 A	*	3 A	:
0 B	K	1 B	[	2 B	+	3 B	;
0 C	L	1 C	＼	2 C	,	3 C	<
0 D	M	1 D	[	2 D	−	3 D	=
0 E	N	1 E	↑	2 E	.	3 E	>
0 F	O	1 F	←	2 F	／	3 F	?

図3-5　VDG内部キャラクタ

次にセミグラフィック時の4モードおよび6モードについて調べます．

N_{60}-BASIC では，6モード(64×48セミグラフィックモード)をサポートしています．次のプログラムを実行して直線の太さをよく見ていて下さい．

```
10 SCREEN 2,2,2:COLOR 1:CLS
20 LINE(0,93)-(256,93),2:LINE(0,0)-(256,192),3
30 FOR I=&HE000 TO &HE1FF:A=PEEK(I):POKE I,A AN
   D &HDF:NEXT I
```

直線が太くなったのが分かると思います．6モードでは1点は4×4ドットでしたが，4モードでは4×6ドットになるために，2ドット分だけ線が太くなったのです．

(5) $\bar{A}$/S(bit6)

アルファニューメリックとセミグラフィックの切り換えに使われます．〝0〞のときにアルファニューメリック(モード1)になり，〝1〞でセミグラフィック(モード2)になります．

(6) $\bar{A}$/G(bit7)

アルファニューメリックとグラフィックの切り換えに用いられます．"0"のときにはアルファニューメリックモード(モード1およびモード2)，"1"でグラフィックモード(モード3，4)になります．

3-3 VRAMのアドレス

3-3-1 アトリビュートアドレスマップ

アトリビュートエリアは，モードに関係なく512バイトに固定されています．

アドレスを表にすると次のようになります．

行＼桁	0	1	2	3	4	5	6	7	8	9		23	24	25	26	27	28	29	30	31
0	X000	X001	X002	X003	X004	X005	X006	X007	X008	X009	········	X017	X018	X019	X01A	X01B	X01C	X01D	X01E	X01F
1	X020	X021	X022	X023	X024	X025	X026	X027	X028	X029	········	X037	X038	X039	X03A	X03B	X03C	X03D	X03E	X03F
2	X040	X041	X042	X043	X044	X045	X046	X047	X048	X049	········	X057	X058	X059	X05A	X05B	X05C	X05D	X05E	X05F
3	X060	X061	X062	X063	X064	X065	X066	X067	X068	X069	········	X077	X078	X079	X07A	X07B	X07C	X07D	X07E	X07F
4	X080	X081	X082	X083	X084	X085	X086	X087	X088	X089	········	X097	X098	X099	X09A	X09B	X09C	X09D	X09E	X09F
5	X0A0	X0A1	X0A2	X0A3	X0A4	X0A5	X0A6	X0A7	X0A8	X0A9	········	X0B7	X0B8	X0B9	X0BA	X0BB	X0BC	X0BD	X0BE	X0BF
6	X0C0	X0C1	X0C2	X0C3	X0C4	X0C5	X0C6	X0C7	X0C8	X0C9	········	X0D7	X0D8	X0D9	X0DA	X0DB	X0DC	X0DD	X0DE	X0DF
7	X0E0	X0E1	X0E2	X0E3	X0E4	X0E5	X0E6	X0E7	X0E8	X0E9	········	X0F7	X0F8	X0F9	X0FA	X0FB	X0FC	X0FD	X0FE	X0FF
8	X100	X101	X102	X103	X104	X105	X106	X107	X108	X109	········	X117	X118	X119	X11A	X11B	X11C	X11D	X11E	X11F
9	X120	X121	X122	X123	X124	X125	X126	X127	X128	X129	········	X137	X138	X139	X13A	X13B	X13C	X13D	X13E	X13F
10	X140	X141	X142	X143	X144	X145	X146	X147	X148	X149	········	X157	X158	X159	X15A	X15B	X15C	X15D	X15E	X15F
11	X160	X161	X162	X163	X164	X165	X166	X167	X168	X169	········	X177	X178	X179	X17A	X17B	X17C	X17D	X17E	X17F
12	X180	X181	X182	X183	X184	X185	X186	X187	X188	X189	········	X197	X198	X199	X19A	X19B	X19C	X19D	X19E	X19F
13	X1A0	X1A1	X1A2	X1A3	X1A4	X1A5	X1A6	X1A7	X1A8	X1A9	········	X1B7	X1B8	X1B9	X1BA	X1BB	X1BC	X1BD	X1BE	X1BF
14	X1C0	X1C1	X1C2	X1C3	X1C4	X1C5	X1C6	X1C7	X1C8	X1C9	········	X1D7	X1D8	X1D9	X1DA	X1DB	X1DC	X1DD	X1DE	X1DF
15	X1E0	X1E1	X1E2	X1E3	X1E4	X1E5	X1E6	X1E7	X1E8	X1E9	········	X1F7	X1F8	X1F9	X1FA	X1FB	X1FC	X1FD	X1FE	X1FF

"X" はページによって異なります．

ページ	RAM 32KB	RAM 16KB
1	8	C
2	E	E
3	C	
4	A	

図3-6 ページとXの値

3-3-2 テキスト・セミグラフィックアドレスマップ

これもアトリビュートと同じで32文字×16行で512バイトになります．

行＼桁	0	1	2	3	4	5	6	7	8	9		23	24	25	26	27	28	29	30	31
0	X200	X201	X202	X203	X204	X205	X206	X207	X208	X209	....	X217	X218	X219	X21A	X21B	X21C	X21D	X21E	X21F
1	X220	X221	X222	X223	X224	X225	X226	X227	X228	X229	....	X237	X238	X239	X23A	X23B	X23C	X23D	X23E	X23F
2	X240	X241	X242	X243	X244	X245	X246	X247	X248	X249	....	X257	X258	X259	X25A	X25B	X25C	X25D	X25E	X25F
3	X260	X261	X262	X263	X264	X265	X266	X267	X268	X269	....	X277	X278	X279	X27A	X27B	X27C	X27D	X27E	X27F
4	X280	X281	X282	X283	X284	X285	X286	X287	X288	X289	....	X297	X298	X299	X29A	X29B	X29C	X29D	X29E	X29F
5	X2A0	X2A1	X2A2	X2A3	X2A4	X2A5	X2A6	X2A7	X2A8	X2A9	....	X2B7	X2B8	X2B9	X2BA	X2BB	X2BC	X2BD	X2BE	X2BF
6	X2C0	X2C1	X2C2	X2C3	X2C4	X2C5	X2C6	X2C7	X2C8	X2C9	....	X2D7	X2D8	X2D9	X2DA	X2DB	X2DC	X2DD	X2DE	X2DF
7	X2E0	X2E1	X2E2	X2E3	X2E4	X2E5	X2E6	X2E7	X2E8	X2E9	....	X2F7	X2F8	X2F9	X2FA	X2FB	X2FC	X2FD	X2FE	X2FF
8	X300	X301	X302	X303	X304	X305	X306	X307	X308	X309	....	X317	X318	X319	X31A	X31B	X31C	X31D	X31E	X31F
9	X320	X321	X322	X323	X324	X325	X326	X327	X328	X329	....	X337	X338	X339	X33A	X33B	X33C	X33D	X33E	X33F
10	X340	X341	X342	X343	X344	X345	X346	X347	X348	X349	....	X357	X358	X359	X35A	X35B	X35C	X35D	X35E	X35F
11	X360	X361	X362	X363	X364	X365	X366	X367	X368	X369	....	X377	X378	X379	X37A	X37B	X37C	X37D	X37E	X37F
12	X380	X381	X382	X383	X384	X385	X386	X387	X388	X389	....	X397	X398	X399	X39A	X39B	X39C	X39D	X39E	X39F
13	X3A0	X3A1	X3A2	X3A3	X3A4	X3A5	X3A6	X3A7	X3A8	X3A9	....	X3B7	X3B8	X3B9	X3BA	X3BB	X3BC	X3BD	X3BE	X3BF
14	X3C0	X3C1	X3C2	X3C3	X3C4	X3C5	X3C6	X3C7	X3C8	X3C9	....	X3D7	X3D8	X3D9	X3DA	X3DB	X3DC	X3DD	X3DE	X3DF
15	X3E0	X3E1	X3E2	X3E3	X3E4	X3E5	X3E6	X3E7	X3E8	X3E9	....	X3F7	X3F8	X3F9	X3FA	X3FB	X3FC	X3FD	X3FE	X3FF

″X″の値はページによって異なります.

ページ	RAM 32KB	RAM 16KB
1	8	C
2	E	E
3	C	
4	A	

図3-7　ページとXの値

3-3-3　グラフィックアドレスマップ

グラフィックモードでは32バイト×192ラインで6144バイトのメモリを使用しています.

ライン	0	1	2	3	4	5	6	7	8	9		23	24	25	26	27	28	29	30	31
0	0200	0201	0202	0203	0204	0205	0206	0207	0208	0209	········	0217	0218	0219	021A	021B	021C	021D	021E	021F
1	0220	0221	0222	0223	0224	0225	0226	0227	0228	0229	········	0237	0238	0239	023A	023B	023C	023D	023E	023F
2	0240	0241	0242	0243	0244	0245	0246	0247	0248	0249	········	0257	0258	0259	025A	025B	025C	025D	025E	025F
3	0260	0261	0262	0263	0264	0265	0266	0267	0268	0269	········	0277	0278	0279	027A	027B	027C	027D	027E	027F
4	0280	0281	0282	0283	0284	0285	0286	0287	0288	0289	········	0297	0298	0299	029A	029B	029C	029D	029E	029F
5	02A0	02A1	02A2	02A3	02A4	02A5	02A6	02A7	02A8	02A9	········	02B7	02B8	02B9	02BA	02BB	02BC	02BD	02BE	02BF
6	02C0	02C1	02C2	02C3	02C4	02C5	02C6	02C7	02C8	02C9	········	02D7	02D8	02D9	02DA	02DB	02DC	02DD	02DE	02DF
7	02E0	02E1	02E2	02E3	02E4	02E5	02E6	02E7	02E8	02E9	········	02F7	02F8	02F9	02FA	02FB	02FC	02FD	02FE	02FF
8	0300	0301	0302	0303	0304	0305	0306	0307	0308	0309	········	0317	0318	0319	031A	031B	031C	031D	031E	031F
9	0320	0321	0322	0323	0324	0325	0326	0327	0328	0329	········	0337	0338	0339	033A	033B	033C	033D	033E	033F
10	0340	0341	0342	0343	0344	0345	0346	0347	0348	0349	········	0357	0358	0359	035A	035B	035C	035D	035E	035F
11	0360	0361	0362	0363	0364	0365	0366	0367	0368	0369	········	0377	0378	0379	037A	037B	037C	037D	037E	037F
12	0380	0381	0382	0383	0384	0385	0386	0387	0388	0389	········	0397	0398	0399	039A	039B	039C	039D	039E	039F
13	03A0	03A1	03A2	03A3	03A4	03A5	03A6	03A7	03A8	03A9	········	03B7	03B8	03B9	03BA	03BB	03BC	03BD	03BE	03BF
14	03C0	03C1	03C2	03C3	03C4	03C5	03C6	03C7	03C8	03C9	········	03D7	03D8	03D9	03DA	03DB	03DC	03DD	03DE	03DF
15	03E0	03E1	03E2	03E3	03E4	03E5	03E6	03E7	03E8	03E9							03FC	03FD	03FE	03FF
⋮					⋮						⋮					⋮				
181	18A0	18A1	18A2	18A3	18A4	18A5	18A6	18A7	18A8	18A9	········	18B7	18B8	18B9	18BA	18BB	18BC	18BD		
182	18C0	18C1	18C2	18C3	18C4	18C5	18C6	18C7	18C8	18C9	········	18D7	18D8	18D9	18DA	18DB	18DC	18DD	18DE	18DF
183	18E0	18E1	18E2	18E3	18E4	18E5	18E6	18E7	18E8	18E9	········	18F7	18F8	18F9	18FA	18FB	18FC	18FD	18FE	18FF
184	1900	1901	1902	1903	1904	1905	1906	1907	1908	1909	········	1917	1918	1919	191A	191B	191C	191D	191E	191F
185	1920	1921	1922	1923	1924	1925	1926	1927	1928	1929	········	1937	1938	1939	193A	193B	193C	193D	193E	193F
186	1940	1941	1942	1943	1944	1945	1946	1947	1948	1949	········	1957	1958	1959	195A	195B	195C	195D	195E	195F
187	1960	1961	1962	1963	1964	1965	1966	1967	1968	1969	········	1977	1978	1979	197A	197B	197C	197D	197E	197F
188	1980	1981	1982	1983	1984	1985	1986	1987	1988	1989	········	1997	1998	1999	199A	199B	199C	199D	199E	199F
189	19A0	19A1	19A2	19A3	19A4	19A5	19A6	19A7	19A8	19A9	········	19B7	19B8	19B9	19BA	19BB	19BC	19BD	19BE	19BF
190	19C0	19C1	19C2	19C3	19C4	19C5	19C6	19C7	19C8	19C9	········	19D7	19D8	19D9	19DA	19DB	19DC	19DD	19DE	19DF
191	19E0	19E1	19E2	19E3	19E4	19E5	19E6	19E7	19E8	19E9	········	19F7	19F8	19F9	19FA	19FB	19FC	19FD	19FE	19FF

実際に使用する場合はページによってアドレスの補正を行なう必要があります.

例　**RAM32KB** でページ2のとき

実際のアドレス＝**E000H**＋ 表のアドレス 　　補正値

ページ2＝**E000H**

ページ3＝**C000H**

ページ4＝**A000H**

3-4 キャラクタ・ジェネレータ

キャラクタ・ジェネレータ ROM は，アルファニューメリックモードでは CG(キャラクタ・ジェネレータ)として，グラフィックモード時では，グラフィックパターン ROM として使用されています．この CG ROM は OUT 命令によって ROM のバンクを切り換えて読み出すことが可能で，CG ROM の1文字あたり16バイト分ある内の12バイトを使用して1文字を構成しています．たとえば"B"の文字は次のようになっています．

これを出力するプログラムは下記のとおりです．

```
 10 OUT &H93,4:AD=&H6000+16*&H42
 20 PRINT "ADR     76543210  DB"
 30 FOR I=AD TO AD+11:DA=INT(I/256):GOSUB 200:PR
    INT HE$;
 40 DA=I-INT(I/256)*256:GOSUB 200:PRINT HE$;"H "
    ;
 50 A=PEEK(I):DA=A:FOR J=7 TO 0 STEP -1:IF A>=2^
    J THEN COLOR 2:A=A-2^J
 60 PRINT " ";:COLOR 1:NEXT J:PRINT "  ";:GOSUB
    200:PRINT HE$:NEXT I
 70 OUT&H93,5:END
200 REM HEX$
210 HE$="00":HE=INT(DA/16):HF=HE:GOSUB 230
220 HE=DA-(16*HF):GOSUB 230:HE$=RIGHT$(HE$,2):RE
    TURN
230 IF HE>9 THEN HE=HE+55:GOTO 250
240 HE=HE+&H30
250 HE$=HE$+CHR$(HE):RETURN
```

なお，OUT&H93,4 でCG ROM を6000H 番地から読むことができ，OUT&H93,5 で CG を OFF にします．

キャラクタ・フォントを出力するプログラムを紹介しておきます．

```
10 OUT &H93,4:FOR K=0 TO 255:AD=&H6000+16*K
20 DA=INT(AD/256):GOSUB 200:PRINT HE$;:DA=AD-IN
   T(AD/256)*256:GOSUB200
30 PRINT HE$;:DA=K:GOSUB 200:PRINT"h ";HE$;" ";
40 IF K<32 THEN PRINT CHR$(&H14);CHR$(K+&H30);"
   ";:GOTO 60
50 PRINT CHR$(K);"  ";
60 FOR I=AD TO AD+5:DA=PEEK(I):GOSUB 200:PRINT
   HE$;" ";:NEXTI:PRINT
70 FOR I=AD+6 TO AD+11:DA=PEEK(I):GOSUB 200:PRI
   NT TAB(12);HE$;" ";:NEXT
80 PRINT:NEXT K:OUT&H93,5:END
200 REM HEX$
210 HE$="00":HE=INT(DA/16):HF=HE:GOSUB 230
220 HE=DA-(16*HF):GOSUB 230:HE$=RIGHT$(HE$,2):RE
    TURN
230 IF HE>9 THEN HE=HE+55:GOTO 250
240 HE=HE+&H30
250 HE$=HE$+CHR$(HE):RETURN
```

CG は完全にデコードされていないために, 7000～7FFFH まではイメージが出力されます.

3-5 ページの切り換え

出力ポートの B0H の bit1, 2 が画面の切り換えのポートになっており, この値を変えることによってページを切り換えることができます.

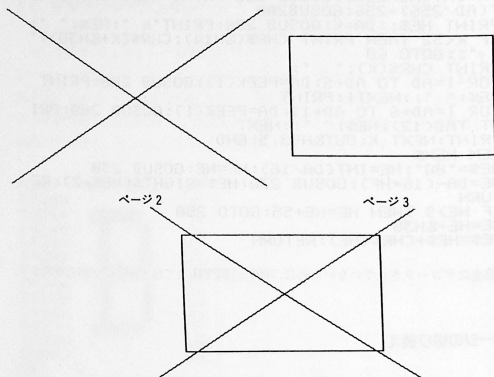
bit2	bit1	VRAMアドレス	RAM32KB時のページ	RAM16KB時のページ
0	0	C000H	3	1
0	1	E000H	2	2
1	0	8000H	1	
1	1	A000H	4	

図3-8　ページの切り換えポート

出力ポートに出力するだけで画面を切り換えられますので, これをうまく使えば画面の擬似重ね合わせができます.

プログラム例

```
10 SCREEN 4,2,2:CLS
20 LINE(0,0)-(256,192):LINE(256,0)-(0,192)
30 SCREEN 4,3,3:CLS
40 LINE(30,30)-(220,160),,B
50 OUT&HB0,0:OUT&HB0,2:GOTO50
```

実行例　このように画面の重ね合わせが可能

ページ2とページ3を50行で切り換えていますので少しチラつきますが，×と□が重なって見えるはずです(注：このプログラムを実行するにはページの指定が3または4になっていなければなりません).

ただし，この重ね合わせは，グラフィックどうし，またはアルファニューメリックモードどうしならば問題はありませんが，グラフィックとアルファニューメリックの組み合わせでは，画面の同期がとれません．(逆にこれをゲーム等に利用するとおもしろいと思いますが).

3-6 表示期間

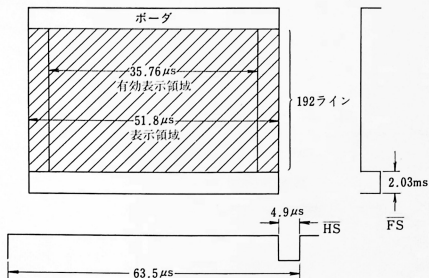
画面にノイズを出さないようにするために，表示期間中はDMA(ダイレクト・メモリ・アクセス)を行ないCPUを停止させており，帰線消去の間のみ，メモリのリード，ライトを許可しています．このために，CPUにかなりのロスタイムが生じています．

※斜線部分でCPUが停止する

図3-9 画面表示期間

第4章　キー入力

第4章　キー入力

4-1 ファンクションキー

4-1-1　メモリの格納状態

ファンクションキーの初期設定は次のようになっています.

```
COLOR          SCREEN
CLOAD"         CSAVE"
GOTO           PRINT
LIST           PLAY
RUN            CONT
```

これらの内容は，ワークエリア上の FB3DH 番地から FB8CH 番地に格納されています.

```
FB3D 43 4F 4C
FB40 4F 52 20 00 00 43 4C 4F 41 44 22 00 00 47 4F 54
FB50 4F 20 00 00 00 4C 49 53 54 20 00 00 00 52 55 4E
FB60 0D 00 00 00 00 53 43 52 45 45 4E 20 00 43 53 41
FB70 56 45 22 00 00 50 52 49 4E 54 20 00 00 50 4C 41
FB80 59 20 00 FF FF 43 4F 4E 54 0D 00 00 00 FF FF 13
```

このように1つのファンクションキーに対して8バイト使用されており，8文字に満たない場合はエンドマークとして「00」が書き込まれます.

8文字で完全に管理されているために，PC-8001のように隣のあいているファンクションキー領域を使用し，9文字以上定義することはできません.

4-1-2　ROM内の格納状態

ファンクションキーの内容はROM内の0167H番地から017AH番地に中間言語＋キャラクタ(1文字)の2バイトで入っており，初期設定のときにFA33～FA46Hに転送されます.

そして中間言語をキーワードに変換してファンクションキー・バッファに入れます.

```
F 1   0167   9A 20   COLOR+sp
F 2   0169   A3 22   CLOAD+"
F 3   016B   88 20   GOTO+sp
F 4   016D   97 20   LIST+sp
F 5   016F   89 0D   RUN+cr
F 6   0171   9F 20   SCREEN+sp
F 7   0173   A4 22   CSAVE+"
F 8   0175   95 20   PRINT+sp
F 9   0177   A7 20   PLAY+sp
F10   0179   96 0D   CONT+cr
```

	RAM	ROM
F 1	FA33	0167
F 2	FA35	0169
F 3	FA37	016B
F 4	FA39	016D
F 5	FA3B	016F
F 6	FA3D	0171
F 7	FA3F	0173
F 8	FA41	0175
F 9	FA43	0177
F 10	FA45	0179

図4-1　キー内容の格納アドレス

プログラム中でファンクションキーのイニシャライズをするときは，KEY コマンドよりは機械語を使う方が簡単です．その方法は第9章で述べます．

4-1-3　KEY LIST

画面にはファンクションキーの内容表示が5文字までしかなされません．N_{60}-BASIC では内容をすべて表示する KEY LIST の命令がありませんので，ファンクションキーの内容を見たいときはそれぞれのキーを押すしか方法がありません．

BASIC でキーの内容を表示するプログラムを下記に示します．

```
 10 I=0
 20 K=I:GOSUB 100
 30 PRINT TAB(10);:K=I+5:GOSUB 100:PRINT
 40 I=I+1:IF I=5 THEN END
 50 GOTO 20
100 FOR J=0 TO 7
110 A=PEEK(&HFB3D+(8*K+J))
115 IF A=0 THEN 140
120 IF A<32 THEN A=32
130 PRINT CHR$(A);:NEXT J
140 RETURN
```

4-1-4　内容の定義の仕方

ファンクションキーの定義は以下のように行ないます．

ファンクションキー定義の例

```
key1,"ABCDE"
Ok
key 2,"12345"+chr$(13)
Ok
a$="AAAAAA":key3,a$
Ok
key 4,chr$(28)+chr$(29)+chr$(30)
+chr$(31)
Ok
key5,chr$(34)+"UFO"+chr$(34)+chr
$(13)
Ok
```

```
1 ABCDE 12345 AAAAA       "UFO"
```

ファンクションキーにはコントロールコードも定義できますが，画面のウィンドウには表示されません．

```
key 1,chr$(18)+chr$(18)+chr$(18)
+chr$(18)
Ok
key 2,chr$(30)+chr$(30)+chr$(30)
+chr$(30)
Ok
key 3,chr$(0)+chr$(1)+chr$(2)+ch
r$(3)
Ok
key 4,"load"+chr$(13)
Ok
key 5,"list."+chr$(30)+chr$(30)
Ok
```

```
1                   load  list.
```

PC-6001 のファンクションキー

```
1                   load  list.
```

PC-8001 のファンクションキー

4-1-5 キーポインタとファンクションキーフラグの使い方

ファンクションキーフラグを使うと，ファンクションキーが押されたのと等価の働きをします．この機能を使うと，いろいろとおもしろいことができます．

```
10 KEY 1,"LIST"+CHR$(13)
20 POKE &HFB8D,&H3D:POKE &HFB8E,&HFB
30 POKE &HFA32,5

RUN
Ok
LIST

10 KEY 1,"LIST"+CHR$(13)
20 POKE &HFB8D,&H3D:POKE &HFB8E,
&HFB
30 POKE &HFA32,5
Ok
```

キーポインタは FB8DH と FB8EH で，キーフラグは FA32H です．

キーフラグには，キー内容の文字数を設定します．上記のプログラムでは，ファンクション キーの内容が，"LIST"+0DH の 5 文字なので30行で POKE する値が 5 になっています．

これから分かるように，キーフラグに設定できる文字数は最大255文字になります．

ただし，文字列の中に00が入っていると，そこで文字列の終了と解釈されます．

4-2 キー入力ステートメント

N$_{60}$-BASIC には，キー入力ステートメントが INPUT と INKEY$ の2種類しかありません．この2つを比較してみましょう．

			INPUT		INKEY$
		書　式	INPUT A,INPUTA$		A$＝INKEY$
入力表示		プロンプト（"　　"）	可能		不可
入力表示		入力指示の？表示	有		無
入力表示		カーソル表示	有		無
入力表示		入力時のカーソル移動	有		無
入力表示		エコーバック	有		無
入力表示		入力待ち	待つ		待たない
キー入力	入力文字の種類	英数，カタカナ ヒラガナ，ファンクションキー，グラフィックキー	A	A$	可
キー入力	入力文字の種類		数字のみ	可	
キー入力	入力文字の種類	特殊文字	カンマは変数の区切り		可
キー入力	入力文字の種類	コントロールコードの入力	不可		可
キー入力		入力文字数	39桁以内	71文字以内	1文字
キー入力		実　行	RET キー		自動的
キー入力		入力文字なしで RET キー	前に入力した値		CHR$(13)
キー入力		使用できる画面モード	1または2		全モード可
入力中断		STOP キー	中断		中断
入力中断		CTRL ＋ C	中断		中断
入力中断		BREAK時のBEEP音	音がでる		音がでる
入力中断		CONTによるプログラム再開	可		可

図4-2　キー入力比較

4-2-1　INPUT

INPUT 命令はテキストモードでしか使用できません．たとえばページ2でモード3のときに INPUT 命令があると，ページ1，モード1でキー入力になります．

```
例：10　SCREEN 3, 2, 2
    20　INPUTA$
```

そのためモード3，4においてINPUTを使うようなプログラムはつくれないことになりますので思考型のゲーム等をつくる場合は注意が必要です．

これを解決する方法としては INKEY$ を使って複数の文字を入力するようにプログラムを組めばよいでしょう．

4-2-2 INKEY$

INKEY$ コマンドを使うと，キー入力を待たないで次の処理に進めますのでリアルタイム処理向きといえます．

また，IF 文で判断させて，文字に対応した行に飛ばすのが一般的な利用方法です．

例 トランプゲームのとき

```
1000 PRINT "ト゛ノ カート゛ ヲ ステマスカ (1-5)"
1010 A$=INKEY$:IF A$="1" THEN 1100
1020 IF A$="2" THEN 1200
1030 IF A$="3" THEN 1300
1040 IF A$="4" THEN 1400
1050 IF A$="5" THEN 1500
1060 GOTO 1010
```

INKEY$ を使ってモード3，4で文字列を入力する方法を示します．

INKEY$による文字列の入力

```
 10 SCREEN 3,2,2:CLS
 20 A$=""
 30 B$=INKEY$:IF B$="" THEN 20
 40 B=ASC(B$):IF B=13 THEN 100
 50 A$=A$+B$:PRINT B$;
 60 GOTO 20
100 END
```

4-2-3 STICK, STRIG

PC-8001のリアルタイムゲームのほとんどは，テンキーとスペースキーを使って遊ぶようになっています．PC-8001 の場合は，I/O ポートを INP 命令で調べることに よって，簡単に，押されたキーを知ることができますが，PC-6001 はキーボードをサブ CPU が管理しているためにこの方法が使えません．それではゲームで遊ぶとき不便なので STICK と STRIG の命令があります．

STICK はジョイスティック(カーソルキー)の方向を調べる命令で，STRIG は発射ボタン(スペースキー)が押されているか調べます．ジョイステックは2人まで遊べるようにと，2個同時につながるようになっています．

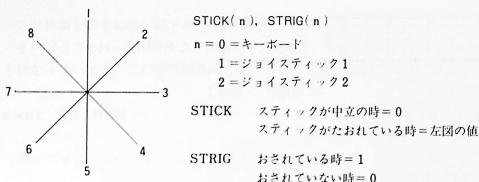



図4-3　ジョイスティックの方向とSTICK・STRIG

STICK, と STRIG を機械語で使用するときの方法を説明します.

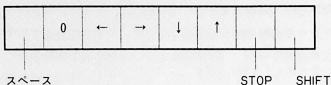
```
0000 3E00        STICK: LD   A,0
0002 CD3922             CALL 2239H
0005 CD4107             CALL 0741H
0008 7B                 LD   A,E
0009 C9                 RET

0000 3E00        STRIG: LD   A,0
0002 CD5922             CALL 2259H
0005 CD4107             CALL 0741H
0008 7B                 LD   A,E
0009 C9                 RET
```

Acc(アキュムレータ)にスティックの番号(00~02H)をセットして,サブルーチンをコールします.データがFAC(浮動小数点アキュムレータ)にセットされていますので,0741Hをコールして FACの値をDEレジスタペアにセットし, Eの値をAccに移します. この部分は第9章でも説明してあります.

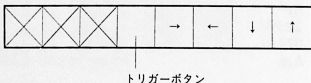
また, 実際にゲームなどで使用する場合は, スティックの方向より, どのスイッチが押されているかが分かったほうが使いやすいでしょう. その方法を簡単に説明します.

●スティック0

```
CALL  1061H
RET
```

結果が次の図のようにAccに返ってきます.

各キーを押しているとその
ビットが1になる．

●スティック1，2

```
LD  A, STICKNO ; スティックの番号1または2
CALL  1CA6H
RET
```

4-2-4 キーバッファ

キースキャンはサブ CPU が担当しており，キー入力があるたびに割り込みをかけてメイン CPU に送ります．受け取ったデータを一時キーバッファに格納し，INPUT 待ちになったときにキーバッファより取り出します．そのため，メイン CPU が他の処理をしている間に次の処理をキー入力することができます．

先行キー入力の実験

```
10 CLEAR 300:CLS
20 FOR I=1 TO 400:LOCATE0,0:PRINT I:NEXT I
30 INPUT A$:PRINT A$
```

上のプログラムを実行すると画面の左上に数字が表示されます．このときに何かキーを押して下さい．数字が400までカウントすると30行の INPUT 命令を実行します．このときに先ほどキー入力した文字が画面に表示されます．この先行入力機能は便利なようですが，ゲームなどでは非常に使いづらいものとなります．

なぜなら，ゲーム等でキー操作を行なっている場合，たいていの人はしばらくキーを押したままにしますので，キーリピートの機能が働き，そのキーのデータが，連続してキーバッファに取り込まれてしまうのです．

INKEY$ を使っている，ポーカー，マージャンなどのいくつかのゲームの中には，この影響をもろに受けて，同じカードやパイを連続して捨ててしまうなどの現象を起こすものがあります．

この現象を防ぐには，以下の方法(通称キーバッファ殺し)を使うとよいでしょう．

キーバッファは，FBB9H より，64バイト分あり，FB8F～FB94H がこのバッファの管理を行なっています．この管理部分のポインタをクリアすれば良いわけです．

```
exec &h1058
```

次のプログラムは，キーバッファがクリアされる例です．

```
10 CLEAR 300:CLS
20 FOR I=1 TO 400:LOCATE0,0:PRINT I:NEXT I
25 EXEC&H1058
30 INPUT A$:PRINT A$
```

4-3 コントロールキー

キャラクタコードの 00～1FH はコントロールコードになっており，画面，プリンタ，RS-232C 等に出力すると，特別の動作を行ないます．たとえば 0CH(10進で12)はキャラクタ・コード表では"秒"の文字になっていますが，PRINT CHR$(12)を実行すると，画面に秒の文字を表示せずに画面がクリアされます．

PC-8001 はコントロールコードのシンボルを表示することができますが，PC-6001 では，その部分がグラフィックキャラクタになっているために表示することができません．

PC-8001 のシンボルキャラクタ

　SH SX EX ET EQ AK BL BS HT LF HM CL RS SO SI
DE D1 D2 D3 D4 NK SN EB CN EM SB EC → ← ↑ ↓

PC-6001 のグラフィックキャラクタ

　月 火 水 木 金 土 日 年 時 小 分 秒 百 千 万
π ┴ ┬ ├ ┼ ┤ │ ─ ┌ ┐ └ ┘ × 大 中 小

PC-8001 と PC-6001 のコントロールコードの機能は，PC-6001 で INS(インサート)の機能が変わったため，行を 2 つに分ける CTRL + J の用法が変わった他は，ほぼ同じになっています．

これらのコントロールコードは，前述の PRINT CHR$ の書式で使う他に，CTRL キーと，アルファベットの組み合わせで入力することができます．

16進	10進	シンボル	シンボルの意味	対応するキー	機　　能
0 0	0		null		
0 1	1	S H	Start of Heading(ヘッディング開始)	CTRL + A	
0 2	2	S X	Start of Text(テキスト開始)	〃 + B	カーソルを1項目ごとに左へ移す.
0 3	3	E X	End of Text(テキスト終了)	〃 + C	STOPと同意(プログラムの実行を中止して, N60-BASICコマンドモードに戻る.
0 4	4	E T	End of Transmission(伝送終了)	〃 + D	
0 5	5	E Q	Enquiry(問合わせ)	〃 + E	カーソル位置から後ろをまっ消する.
0 6	6	A K	Acknowledge(肯定応答)	〃 + F	カーソルを1項目ごとに右にずらす.
0 7	7	B L	Bell(ベル, ブザー)	〃 + G	内蔵のブザーを0.5秒間鳴らす.
0 8	8	B S	Back Space(後退)	〃 + H	DEL カーソルのすぐ左の1文字を除する.
0 9	9	H T	Horizontal Tabulation(水平タブ)	〃 + I	8文字ごとの水平タイプ。
0 A	10	L F	Line Feed(改行)	〃 + J	行を2つに分ける. カーソルを次の行へ移す.
0 B	11	H M	Home(VT)Vertical Tabulation(垂直タブ)	〃 + K	カーソルをホームポジション画面左上に戻す.
0 C	12	C L	Clear(FF)Form Feed(改頁)	〃 + L	画面を消去してカーソルをホームポジションに戻す.
0 D	13	C R	Carriage Return(復帰)	〃 + M	カーソルを次の行の先頭に移す.
0 E	14	S O	Shift-out(シフトアウト)	〃 + N	
0 F	15	S I	Shift-in(シフトイン)	〃 + O	
1 0	16	D E	Data Link Escape(伝送制御拡張)	〃 + P	
1 1	17	D 1	Device Control1(装置制御1)	〃 + Q	
1 2	18	D 2	Device Control2(装置制御2)	〃 + R	INS
1 3	19	D 3	Device Control3(装置制御3)	〃 + S	
1 4	20	D 4	Device Control4(装置制御4)	〃 + T	
1 5	21	N K	Negative Acknowledge(否定応答)	〃 + U	1行消去
1 6	22	S N	Synchronous idle(同期信号)	〃 + V	
1 7	23	E B	End of Transmission Block(伝送ブロック終了)	〃 + W	
1 8	24	C N	Cancel(取消し)	〃 + X	
1 9	25	E M	End of Medium(媒体終端)	〃 + Y	
1 A	26	S B	Substitute(文字置換)	〃 + Z	
1 B	27	E C	Escape(拡張)	ESC	実行の一時中断
1 C	28	→	(FS)File Separator(ファイル分離)	→	カーソルを1つ右へ移す.
1 D	29	←	(GS)Group Separator(グループ分離)	←	カーソルを1つ左へ移す.
1 E	30	↑	(RS)Record Separator(レコード分離)	↑	カーソルを上の行へ移す.
1 F	31	↓	(US)Unit Separator(ユニット分離)	↓	カーソルを下の行へ移す.

図4-4

第5章　サウンド機能

第５章　サウンド機能

5-1　PLAY 命令の基礎

サウンド機能は PC-6001 を特徴づける機能の１つです．この章ではこのサウンド機能について解説しますが，その初めとして PLAY 命令について説明しましょう．

N_{60}-BASIC の PLAY 命令は音楽を演奏するための命令で，大変分かりやすい音楽用言語(MML：Music Macro Language)を使って音程，音量，音長等のデータを表記します．

音の高さは２種類の表記方法があり，第１の方法は音階名(C，D，E，F，G，A，B)に On，およびシャープ(#または＋)・フラット(－)を併用する方法，もう１つは Nn を使う方法です．

まず音階名による表記とは，出したい音の音階名をＡ～Ｇのアルファベットで列記する方法です．図５－１を参考にしながら次の命令を実行してみてください．

```
PLAY "CDEFGABC"
```

いかがですか？　ドレミファソラシドと演奏しましたね．ところが最後のドは最初のドと同じ高さの音が出てしまいました．これはオクターブの指定をしなかったために起きたこと

図5-1

です．PLAY 命令では，C～Bまでを1オクターブとして区切り，計8オクターブの音域をカバーしています．そしてO1～O8 を MML 群に付け加えることによって，そのオクターブを指定することができます．オクターブの指定は次に新たにオクターブを指定するまで保持されます．それでは先ほどのものにオクターブの指定を付けてみます．

```
PLAY "O4CDEFGABO5C"
```

今度は正しく1オクターブ上のドが出ました．

今度は半音を出してみましょう．半音を出す場合は，半音にしたい音階名の後にシャープの記号(#か＋)やフラットの記号(－，マイナス記号)を付ければよいのです．試しに音を出してみましょう．

```
PLAY "CC+D"
```

ドとレの音の間にド#が出ました．これは次のように書いても同じです．

```
PLAY "CC#D"
PLAY "CD-D"
```

上の例は PLAY 命令中では＋と#が同じ意味をもつためです．下の例はド#とレ♭が同じ音だからです．分からない人は音楽の本を見てください．ですからミ#つまりＥ＋はファ，つまりＦと同じ音になりますし，ド♭つまりＣ－はシ，つまりＢと同じ音になります．ただし，この場合，本来の音より1オクターブ上のC－が出ます(このあたりはある程度，音楽の基礎知識が必要ですから，音楽の苦手な方は読みとばしてもかまいません)．

以上のような方法によって音階表記で，8オクターブの音域をカバーすることができるのです．

それではもう1つの方法，つまり Nn による表記を説明します．これは，最低オクターブのCの音をN1，C#をN2，DをN3というように，音程が半音(100セント)上昇するごとに番号を1つ増やして，最高オクターブのBの音まで，Nの後ろに数字をかいて音の高さを指定する方法です．この方法も試してみましょう．

```
PLAY"N25N27N29N30N32N34N36N37"
```

どうですか？　ドレミファソラシドと演奏しましたね．この方法では，オクターブや半音の指定なども同様にできるので，場合によっては便利なこともありますが，慣れないと使いにくいかもしれません．

これで音程は自由に設定できるようになりました．次は長さを決めてみましょう．

音の長さは基本的には音程を示すコード(Cなど)の後にその音の長さを示す1～64の整数を付け加えることによって設定できます．その数値はたとえば4分音符なら4，32音符なら

32というふうに長さの逆数表示ですから，従来の他のMMLに比べて非常に使いやすくなっています．次の例を試してください．

```
PLAY"C4D8E16F32"
```

音の長さが変わったのがお分かりいただけたと思います．

もしここで音の長さの数値をつけないと，自動的にデフォルト値の長さに設定されます．

デフォルト値は電源投入時には4に設定されていますから，普通は4分音符になります．このデフォルト値も変更することができ，そのMMLはLnという形をとります．たとえばL8を指定すれば，とくに長さの指定のない音符はすべて8分音符になります．また，このデフォルト値もオクターブ指定と同様に，次に指定するまでは，その値が保持されます．次に例を示します．

```
PLAY"L8CDE16F16GAL2B"
```

音を休止したい場合には休符記号であるRを入れます．休符の長さは音の長さと同じようにR4で4分休符，R2で2分休符となります．ただし，後の数字を省略した場合は，Lで指定した長さにはならず，すべて4分休符となります．次の例を試してください．

```
PLAY"L2CR16CR8CR4CRC"
```

次に符点音符の出し方を説明します．符点音符を出す場合は実に簡単で，符点音符にしたい音のコードの後に〝.″(ピリオド)をつけるだけです．たとえばL8CDE.とすればEの音だけが符点8分音符，C2.とすればCの符点2分音符が出ます．

これで音の長さも自由に設定できるわけですが，音程をNnの形で設定した場合，音の高さを示す数字の後に続けて音の長さの数字をかくと，数字が3桁並んでMMLが判別できなくなるので，音程のコードの後に〝;″(セミコロン)を付けてそれに続けて音の長さのコードを書きます．たとえばN36;8という具合です．

それでは次に曲のテンポの設定をしましょう．実際の音楽では，同じ4分音符でも曲のテンポによってその長さが違います．N_{60}-BASICのMMLでは，テンポの設定のためにはTnという方法を使います．たとえばT90とすれば♩=90，つまり1分間に4分音符が90回演奏できるテンポになります．ちなみに電源投入時にはT120に設定されています．またテンポの指定も一度指定すると新しく指定するまではその値が保持されます．

以上の方法を用いて，一声の音楽は一応演奏できるようになりました．それでは次に和声を出してみます．

PC-6001はGI社のAY-3-8910というサウンド用LSIを用いて音を発生させています．このLSIの特徴の1つに同時に3チャネルの音を別個に設定できるということが挙げられます．N_{60}-BASICのMMLもこの特徴を活かして簡単に3つまでの和声のプログラミングが可能になっています．

例としてC，つまりドミソの和音をつくってみましょう．

```
PLAY"O4C","O4E","O4G"
```

上の例でもわかるように，各チャネルで演奏したい音を順に並べて書いて，その間を，(カンマ)で区切ると，その音を一斉に演奏します．これは別に3つでなくても2つだけ使うこともできます．

```
PLAY"C","G"
```

また，テンポ(T)，オクターブ(O)，長さ(L)等の指定は，各チャネル別に行なうことができます．各自試してみてください．

これで3和声の音楽はプログラムできるようになったわけですが，これだけでは大変味気ない演奏になってしまいます．そこで，次に，音楽に変化を付ける方法を考えてみましょう．

音楽に変化を付けるための方法としては，1)音色を変える　2)音量，つまり音の大きさを変える，等が考えられます．PC-6001の場合，音色に変化を付けることは，基本的には不可能ですから，ここでは音の大きさに変化を与える方法を説明します．

N_{60}-BASICのMMLには音量の変化を付ける方法として2種類の方法が準備されています．1つはVnによる方法，もう1つはSlとMnによる方法です．

まずVnによる方法とは，Vの後に0から15の整数を付けることによって，ボリューム，すなわち音量を変化させる方法です．V0を指定すると無音，それからV1，V2と数字を大きくするに応じて音も大きくなり，V15で最大になります．次の例を試してください．

```
PLAY"V1CV5EV9G"
```

音の大きさが変化しました．このVの指定もTやL等と同様に，1度指定すると次に指定するまで変わりませんし，チャネルごとに別個に指定することが可能です．また電源投入時にはV8が設定されます．ちなみにVの数字と音声出力電圧の関係は指数的に変化します．これは，人間の耳の刺激に対する反応が対数的になっているのを補正するためだと思われます．

では次にSlとMnによる，エンベロープパターンの変化のさせ方について説明します．エンベロープパターン，つまり包絡線形状とは，シンセサイザに興味のある方は御存知でしょうが，簡単に言えば，音の大きさの時間に応じた変化の様子のことを言います．

たとえばギターやピアノは，音を発した瞬間は強い音がして，それから時間を経ると音が弱くなっていきます．またファゴットなどでは弱い音からふわっと立ち上がる感じの音がします．

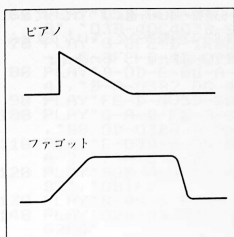



図5－2　エンベロープパターン

この場合のエンベロープパターンを分かりやすく略して書いたものが図5－2です．もちろん実際はこんなに単純ではありません．

PC-6001 に使われているサウンド用 LSI は，このエンベロープをある程度自由に設定できます．このエンベロープ設定のための MML がSとMなのです．Sはエンベロープの形状を，Mはその周期を指定するものです．

S	エンベロープパターン
0,1,2,3,9	
4,5,6,7,15	
8	
10	
11	
12	
13	
14	t

図5－3

具体的には，エンベロープの形状は図5－3に示される8種類の中から選ぶようになっており，表に示された数字をSの後に付記して使います．数字は0～15の整数でなくてはなりません．周期は任意に設定可能で表5－3中のtの長さから公式で決定されます．その公式は

$$M=\frac{1996750\times t}{256} \quad t：周期(秒)$$

という式です．

たとえば周期を2秒に設定したければ，

$$M=\frac{1996750\times 2}{256}\fallingdotseq 15600$$

ですからM15600を指定すれば良いわけです．この数字は1から65535までの自然数でなくてはなりません．

エンベロープを設定するとき，気を付けなくてはならないことが2つあります．第1に，2つ以上の異なったエンベロープを指定できないということです．つまり和声を出したときにチャネルごとにSまたはMの値を違えることはできないのです．同じエンベロープを指定するか，他のチャネルはエンベロープを指定せずにVで音量を指定するかしなければなりません．もう1つ気を付けるべきことは，同一チャネルにエンベロープの指定，つまりSと，音量の指定，つまりVは同時にできないということです．たとえば，

```
PLAY"S0M1000V8……"
```

という指定をすると，後から指定したVの指定が有効になります．

以上でPLAY命令の基礎の説明を終わります．これで一応の音楽演奏プログラムは作成可能になりました．例として簡単な音楽プログラムを紹介しておきます．

```
 0 REM*************************************
 1 REM* La fille aux cheveux de lin       *
 2 REM*                                   *
 3 REM* Composed by C.Debussy (Jan.1910)  *
 4 REM*                                   *
 5 REM* Arranged & Encoded                *
 6 REM*                by Yellow Panther  *
 7 REM*                                   *
 8 REM*          To my dearest "Butch"    *
 9 REM*************************************
10 PLAY"T60L16S0M1000O5","T60L16V10M1000O3","
   T60L16V10O3M10000"
20 PLAY"D-4.OB-G-E-8G-B-O5D-8OB-G-E-8G-B-G-8G-E
   -","R1R4.B","R1R4.G-"
30 PLAY"G-8FE-M30000D-1M10000E-8G-8","O3B-2","G
   -2"
40 PLAY"A-4.O5D-8OB-8G-B-A-8O5D-8OB-4O5","OF2G-
   4F2","D-2E-8D-4.D4"
50 PLAY"E-2OB-4.B8O5D-4.OB-G-E-8G-B-","B-2B-4.B
   8OD-2E-4","E-2
```

```
 60 PLAY"O5D-8OB-G-E-8G-B-G-8G-E-","D-4E-4D-8O3B
    8","O3B-4OC4O3B-8A-8"
 70 PLAY"G-8FE-D-1R8O3","B-2.B-2","G-4OD-4D-O3BB
    -A-G-8FE-D-4D-O2B"
 80 PLAY"B-OD-E-8G-A-L8BO5D-E-G-4L16","R8OD-2.B-
    4","B-A-O3B2.OG-4"
 90 PLAY"FE-D-4V5D-S0OBB-A-","B-4","G-4O3"
100 PLAY"G-A-B-FE-A-G-D-O3BOF","E-8.FE-8.D-O3B8"
    ,"B8.OD-O3B8.B-A-8"
110 PLAY"E-O3B-A-OD-O3C-8.OD-F-A-BO5D-F-A-","BB-
    A-2.","A-G-F-2."
120 PLAY"B8B-A-G-2B-8A-G-E-4.D-OB","O5E-2.OG-4O3
    B2","OB1F2"
130 PLAY"B-8A-G-E-4.E-C","B4B-2","G-4O2B-8R8B-4"
140 PLAY"O2B-8B-O3CE-FGB-OC8E-C","OE-2O3A-4","O3
    G2F4"
150 PLAY"E-8O3B-OCE-FGB-O5C8E-C","O2B-8.O3CE-FGB
    -OC8E-C","G2OF4"
160 PLAY"E-8OB-O5CE-FG-B-O6E-4.","G8O3B-OCE-FG-B
    -B4.","O4B-2O5B4."
170 PLAY"L24D-E-D-O5B-8A-8L16V5A-S0G-FE-L24","O5
    A-8G-8B4.O","F8E-2O"
180 PLAY"E-8D-E-D-L16OB-8A-4G-A-","B8A-8G-8A-4O3
    G-OF","G-8F8E-8C4B8"
190 PLAY"L8B-.G-16E-G-B-O5","L8E-.D-16O3B-OD-E-"
    ,"L8O3B-.G-16E-G-B-O"
200 PLAY"D-OB-G-E-4G-4B-.","G-E-D-O3B-4B4OE-.","
    D-O3B-G-E-4G-4B-."
210 PLAY"G-16B-O5D-E-G-B-G-","D-16E-G-B-O5D-E-D-
    ","G-16B-OD-E-G-B-G-"
220 PLAY"OB-4D-4R2O6D-4V4D-4.S0L16","O3C-4A-4OG-
    1G-","O3G-4F4OE-1E-"
230 PLAY"O5B-G-E-8G-B-O6D-8O5B-G-E-8G-B-G-8G-E-"
    ,"G-G-1","E-E-1"
240 PLAY"LG-FE-D-.OL8BB-A-L16","B-2.D-2.L16","G-
    1.L16O3"
250 PLAY"G-A-B-FE-A-G-D-O3BOF","E-8.FE-8.D-O3B8"
    ,"B8.OD-O3B8.B-A-8"
260 PLAY"E-O3B-A-OD-E-O3B-BO","BB-A-8.B-B","A-G-
    F8.G-A-"
270 PLAY"D-E-O3B-A-O","B8B-A-","A-8G-F-"
280 PLAY"D-E-O3B-BOG-A-BO5D-E-G-A-","A-8OE-8D-E-
    G-A-BO5D-E-","F8G-A-2"
290 PLAY"L12BO6D-E-.","L12G-A-BB","A-4A-24"
300 PLAY"S0M60000OG-1.","S0O6R16D-1.","S0O5R32B-
    1."
```

5-2 SOUND命令

これまでPLAY命令を使った簡単なプログラミングテクニックを紹介してきました．しかし PLAY 命令では音の高さの変化をなめらかにしたり，ノイズを発生したりすることはできません．このような効果音を発生するためにはサウンド用 LSI を直接コントロールする必要があります．サウンド用 LSI を BASIC で直接コントロールするための命令が，これから説明する SOUND 命令です．

SOUND 命令は次のような形になります．

SOUND［レジスタ番号］,［データ］

レジスタとは，一種のメモリのようなもので，それぞれに音の高さとか大きさといったデータを入れます．それではこのレジスタについて少し詳しく説明します．

PC-6001 のサウンド用 LSI のレジスタは付録-10のようになっています．

まず［R0, R1］，［R2, R3］，［R4, R5］の3つのレジスタペアは，それぞれA, B, C各チャネルの音の周波数を決定します．R0, R2, R4がそれぞれの微調整，R1, R3, R5 が主調整で，前者は0～255の整数，後者は0～15の整数がデータとして有効になります．そして出力したい周波数から公式を使って，このデータを求めることができます．その公式は次のとおりです．

$$TP=\frac{1996750}{16\times f_T}$$

$$TP=CT\times 256+FT$$

f_T ：出力する周波数 (Hz)
CT ：主調整レジスタの値(R1, 3, 5)
FT ：微調整レジスタの値(R0, 2, 4)

たとえばAの音，440Hz の音を出すときには，

$$TP=\frac{1996750}{16\times 440}\fallingdotseq 284$$

$$284=\underline{1}\times 256+\underline{28}$$

ですからCT＝1, FT＝28となります．したがって，R0 には28, R1 には1を設定すればよいわけです．

次に R6 は，ノイズの周波数を設定します．このLSIはノイズ発生器を1つ内蔵しており，その周波数を設定できるのです．このレジスタは0～31が有効で，次の公式によってその値を決定します．

$$NP = \frac{1996750}{16 \times f_N}$$

f_N：ノイズ周波数(Hz)

NP：［R6］の値

次に R7 です．このレジスタは，ミクサーの役割りをします．このレジスタの各ビットが，それぞれのチャネルのトーンとノイズ，そしてこの LSI を汎用 I／Oポートとして使う場合の入出力の設定を意味します．ただしこれは負論理です．もっとも，この説明では何のことやら分からない人もいるでしょうから，そういう人のために，このチャネルに設定するデータと，各チャネルのトーン，ノイズの出力の様子を表にしたものを示します(図5-4)ので，参考にして下さい．

表中 &H…と書かれたものが16進表示の R7 のデータ，その右が出力されるトーンとノイズ，およびそのチャネルです．たとえば R7 に F1H を設定するとAチャネルはノイズのみ，B，Cチャネルはトーンのみが出力されます．

R8，R9，R10 はそれぞれチャネルA，B，Cの音量設定，PLAY 命令のVにあたるものです．このレジスタは 0 ～15を設定すると，それに応じた音量になりますが，16を設定すると，そのチャネルの音量はエンベロープジェネレータによって支配されます．つまり，そのチャネルはエンベロープのかかった音になるわけです．

R11，R12 は，エンベロープジェネレータの周期を決定するレジスタペアで，この値は PLAY 命令のMにあたるものです．各レジスタとも 0 ～255が有効で，周期からデータを算出します．その値は PLAY のMの式で求めた値から，

$$EP = CT \times 256 + FT$$

EP：Mの値
CT：主調整レジスタの値［R12］
FT：微調整レジスタの値［R11］

という式を使って求めます．

R13 はエンベロープの形状を決定するレジスタで，PLAY 命令のSに相当するもので，このレジスタに値を与えた瞬間からエンベロープの発生が始まります．

ですから，R13以外のレジスタの値をくずさなければ，後は SOUND13，［データ］を実行すれば何度でも同じ音が出るわけです．

また，このデータを 8 や10，11，12，13，14にすると，後は何もしなくても勝手に音が連続して発生されます．その間 CPU は自由に仕事ができますから，これを上手に利用すれば，ゲームのバックグラウンドに音を出すなどということは大変簡単にできるわけです．

R14，R15 はI／Oポートのデータのレジスタですから，音の発生には直接関係ありません．

以上の方法で，各レジスタのデータが決定したら，これらを実際にレジスタにセットすれ

ば音が出るわけです．
たとえばチャネルＡの音量を最大にしたければ，

SOUND 8,15

また，チャネルＢにエンベロープを付けたければ，

SOUND 9,16

＆ＨＦＦ	出力なし	＆ＨＤＦ	Ｃノイズのみ
ＦＥ	Ａトーンのみ	ＤＥ	ＣノイズＡトーン
ＦＤ	Ｂトーン	ＤＤ	Ｂトーン
ＦＣ	ＡＢトーン	ＤＣ	ＡＢトーン
ＦＢ	Ｃトーン	ＤＢ	Ｃトーン
ＦＡ	ＡＣトーン	ＤＡ	ＡＣトーン
Ｆ９	ＢＣトーン	Ｄ９	ＢＣトーン
Ｆ８	ＡＢＣトーン	Ｄ８	ＡＢＣトーン
Ｆ７	Ａノイズのみ	Ｄ７	ＡＣノイズのみ
Ｆ６	ＡノイズＡトーン	Ｄ６	ＡＣノイズＡトーン
Ｆ５	Ｂトーン	Ｄ５	Ｂトーン
Ｆ４	ＡＢトーン	Ｄ４	ＡＢトーン
Ｆ３	Ｃトーン	Ｄ３	Ｃトーン
Ｆ２	ＡＣトーン	Ｄ２	ＡＣトーン
Ｆ１	ＢＣトーン	Ｄ１	ＢＣトーン
Ｆ０	ＡＢＣトーン	Ｄ０	ＡＢＣトーン
ＥＦ	Ｂノイズのみ	ＣＦ	ＢＣノイズのみ
ＥＥ	ＢノイズＡトーン	ＣＥ	ＢＣノイズＡトーン
ＥＤ	Ｂトーン	ＣＤ	Ｂトーン
ＥＣ	ＡＢトーン	ＣＣ	ＡＢトーン
ＥＢ	Ｃトーン	ＣＢ	Ｃトーン
ＥＡ	ＡＣトーン	ＣＡ	ＡＣトーン
Ｅ９	ＢＣトーン	Ｃ９	ＢＣトーン
Ｅ８	ＡＢＣトーン	Ｃ８	ＡＢＣトーン
Ｅ７	ＡＢノイズのみ	Ｃ７	ＡＢＣノイズのみ
Ｅ６	ＡＢノイズＡトーン	Ｃ６	ＡＢＣノイズＡトーン
Ｅ５	Ｂトーン	Ｃ５	Ｂトーン
Ｅ４	ＡＢトーン	Ｃ４	ＡＢトーン
Ｅ３	Ｃトーン	Ｃ３	Ｃトーン
Ｅ２	ＡＣトーン	Ｃ２	ＡＣトーン
Ｅ１	ＢＣトーン	Ｃ１	ＢＣトーン
Ｅ０	ＡＢＣトーン	Ｃ０	ＡＢＣトーン

図５-４

を入力します.

SOUND 命令について理解されたでしょうか？　なんだか，ミュージック・シンセサイザみたいですね．参考のためにこのサウンド用 LSI のブロックダイアグラムを図5-5に示します．

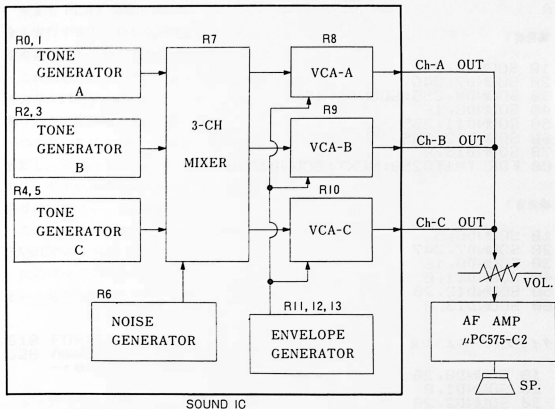



図5-5　AY-3-8910　ブロックダイアグラム

それでは次に，サウンド命令を使ったいろいろな効果音の例を示しておきます．自作のゲーム等に応用してください．

ロケット発射音のような音

```
10 SOUND1,0:SOUND3,0:SOUND12,255:SOUND11,255:SO
   UND6,31:SOUND7,244
15 SOUND8,15:SOUND9,0
20 FOR I=200TO140STEP-0.2:SOUND0,I:NEXT:SOUND9,
   15
25 FORI=140TO40STEP-0.2:SOUND0,I:SOUND2,I+60:NE
   XT:SOUND8,16:SOUND9,16
30 SOUND13,0:FOR I=40TO10STEP-0.2:SOUND0,I:SOUN
   D2,I+60:SOUND6,I-9:NEXT
```

ヘリコプターのような音

```
10 SOUND0,20:SOUND1,0:SOUND2,30:SOUND3,0:SOUND4
   ,0:SOUND5,9:SOUND6,0
20 SOUND7,48:SOUND8,16:SOUND9,4:SOUND10,6:SOUND
   11,100
30 SOUND12,2:SOUND13,12
```

爆発音1

```
10 SOUND6,31
20 SOUND7,246
30 SOUND0,255:SOUND1,15
40 SOUND8,16
50 SOUND11,255
60 SOUND12,255
70 SOUND13,0
80 FOR T=1TO250:NEXT:SOUND13,0
```

爆発音2

```
10 SOUND6,5
20 SOUND7,247
30 SOUND8,16
40 SOUND11,0
50 SOUND12,20
60 SOUND13,0
```

タイプライターのような音

```
 10 SOUND0,20
 20 SOUND1,0
 30 SOUND2,20
 40 SOUND3,20
 50 SOUND4,10
 60 SOUND5,0
 70 SOUND6,8
 80 SOUND7,224
 90 SOUND8,16
100 SOUND9,16
110 SOUND10,16
120 SOUND11,0
130 SOUND12,5
140 SOUND13,0
150 REM ｺﾉｱﾄ SOUND 13,0 ｦ ｼﾞｯｺｳｽﾚﾊﾞ ｿﾉﾀﾋﾞﾆ ｵﾄｶﾞ
    ﾃﾞﾏｽ｡
```

5-3 より高度なテクニック

5-3-1 PLAY のバッファ

今まで PLAY 命令を使ってきて，観察力のある方は気付かれたと思いますが，PLAY 命令を実行すると，音が全部演奏される前に，すでに PC-6001 は Ok を表示して，カーソルが点滅し，入力待ち状態になっています(気付いていなかった人は今すぐ試してください).

これはどういうことかといいますと，PC-6001 は内部に PLAY 命令専用のバッファをもっており，PLAY 命令が与えられると，すべてのデータをバッファに取り込み，あとはタイマ割り込みで取り込んだデータを演奏するしくみになっているのです.

つまり，すべてのデータをバッファに取り込んだ時点で，PLAY 命令の実行は終了したと判断し，次に命令があれば，バッファ内のデータを演奏しながら，その命令の実行に移るのです．ですから PLAY 命令を1つ与えると，音を演奏している途中ですでに次の命令の待ち受け状態になっているわけです.

このことは上手に使えば，タイマ割り込みを使った音の出力と，他の仕事の同時進行が，BASICのみで可能になります.

次のプログラムは PC-6001 用マージャンゲームのキー入力ルーチンの例で，〝ピッポッ〟という音を連続出力しながら捨てハイの入力を INKEY$ で受けます.

```
610 FOR I=1 TO 68
620 A$=INKEY$:IF A$=""THEN NEXT I:PLAY "l16v8o3g
    -r8.e-r8.":GOTO 610
```

ところで，逆にこれが非常にじゃまになることも当然あります.

たとえば音楽の演奏の終了と同時に画面を書き換えたい等という要求があった場合，PLAY 命令の後に続けて画面書き換えの命令を書くと演奏が終了しないうちに画面が書き換えられてしまいます．ですから何らかの方法で演奏終了まで画面操作を待ってやらなければなりません.

まず考えられるのは FOR～NEXT 等を使って時間待ちをする方法です.

しかしこの方法では待ち時間の調整が大変微妙で，音楽演奏の終了と同時に次の命令の実行に移らせるのはむずかしいでしょう.

ではどうすればよいでしょうか.

実は，N_{60}-BASIC のワークエリアには，PLAY 命令中での演奏中のチャネル数，つまり残りのチャネル数を示している部分があって，FD1BH 番地の下位からの3ビットがそれぞれのチャネルに対応して，正論理で演奏中か否かを表わしています．演奏が完全に終了すれば，このアドレスのデータは0になりますからこれを利用すれば演奏終了と同時に次の仕事に移ることができるのです.

次のプログラム例を見てください．音楽が終了すると同時に"END"と画面に表示します.

```
10 CLS
20 PLAY "L2C","L2E","L2G"
30 IF PEEK(&HFD1B)<>0 THEN 30
40 PRINT "END"
```

5-3-2 PLAYに変数を

次にPLAY命令中に変数を使う方法を説明しましょう．

PLAYに変数を使うといえば，まず頭に浮かぶのは文字数にPLAYのデータを代入して使う方法でしょう．たとえば，

```
10 A$="CDE":B$="FGA"
20 PLAY A$+B$
```

といった具合です．

しかし，ここでいう変数は文字変数ではありません．ごく普通の数値変数です．PLAY命令中で使用する数値と言えば，OやV, L, Nなどに与える数値ですが，これに変数を使うわけです．

PLAY命令中に変数を使う場合は次の方法で行ないます．

PLAY"［MML］=［変数名］;"

具体的には，

```
10 FOR NUMERIC=1 TO 96
20 PLAY"N=NUMERIC;32R32"
30 NEXT NUMERIC
```

となります．ここではNUMERICというのが変数名で，この変数をFOR～NEXTループを使って1から96まで変えて，これをMMLの中の〝N″に付けて演奏します．もう1つの例を示しましょう．

```
10 OCTAVE=INT(8*RND(1))+1
20 PLAY "O=OC;C","O=OC;E","O=OC;G"
30 GOTO 10
```

この例ではドミソの和音の高さを乱数で決定しています．10行のOCTAVEが，オクターブを代入する変数です．20行ではOCという変数名になっていますが，変数は最初2文字のみ有効ですからこれでよいのです．

それからこれは余談ですが，VやL, O, Tは，その後の数値を省略してもかまいません．

ただし，省略した場合は，自動的にデフォルト値，つまり電源投入時の値に設定されます．すなわち，

PLAY "TLVOCEG" と,

PLAY "T120L4V8O4CEG"

は, 全く同じ働きなのです. ですからメモリ節約のためにも, この省略は上手に応用しましょう.

5-3-3 PLAYと音色

PC-6001 に使われているサウンド用 LSI のトーン出力は, すべて方形波です. つまり, 1つの音色しか出せません. これではどうしても単調な感じになってしまいます. それを少しでも解決する方法を考えてみましょう.

まず, パーカッションの音を PLAY 命令で出してみましょう.

まず, ハイハットの音です. これは高めのノイズにエンベロープを付ければよさそうです. しかし, PLAY 命令ではノイズを出すことはできません. そこで PLAY と SOUND を組み合わせてみましょう.

```
10 SOUND 6,0
20 SOUND 7,&HF1
30 PLAY "S0M3000C"
```

上のプログラムを試して下さい. なんとなくそれらしい音が出ました. 10行はノイズの音の高さの設定, 20行はAのチャネルにノイズだけを, B, Cのチャネルにトーンだけを出力するように設定しています. あとは PLAY 命令で, 好きなようにノイズが演奏できるわけです. 30行の最後のCは, 別にCでなくてもA～Gであれば何でもかまいません.

次に, ノイズの音を少し変えて, 低いトーンを混ぜて, スネアドラムのような音を出してみましょう(そう聞こえるかどうかはあなたの感性におまかせしますが…).

```
10 SOUND 6,10
20 SOUND 7,&HF0
30 PLAY "S0M3000L16O3E8EEEEE8"
```

今度はチャネルAはトーンも出力されていますから30行のEを変えると当然音が変化します. それから上の2つの例では20行を変えればノイズの出るチャネルが変わるのがお分かりいただけると思います. たとえば, ハイハットの例で,

20 SOUND 7,&HD8

とすれば, 30行は PLAY [A], [B], "S0M3000……" となって, A, Bの部分でメロディーを演奏して, チャネルCはハイハット, となるわけです.

では，次はノイズを全く使わずにつくるパーカッションの音です．次のプログラムを試してください．

```
10 PLAY"S0M10003T150L16C8C+C+D8D+D+E8FFF+8GGG+8
   GGF+8FFE+8EED+8DD"
20 GOTO 10
```

いかがですか？　まだまだできそうですね．

パーカッションはこれでよいとして，メロディーの部分はどうしたらよいのでしょうか．

PLAY 命令で音色に変化をもたせる簡単な方法は，複数のチャネルを使って１つのメロディーを演奏させることです．たとえば，

```
PLAY "S13M100TLOCEG"
```

と，

```
PLAY "S13M100TLOCEG", "S13M100TLO5CEG"
```

とでは大変違った感じの音になります．

チャネルBにチャネルAの２倍の高さ，つまり１オクターブ上の音を重ねてみたのです．

この方法は，チャネル数を１チャネル多く使うという欠点はありますが，大変簡単であり，またチャネルごとに音量を変えたり，混ぜる音の高さの比を変えたりすることによって豊富な変化が得られ，効果的な方法です．各自試して，自分の耳で確かめてください．

さて，この場合に限らず，エンベロープを使うチャネルと，使わないチャネルとで，和声を発生する場合，気を付けねばならないことがあります．これは PLAY 命令の BUG の１つですが，状況によって，エンベロープの周期がくずれてしまうのです．くずれるというのは具体的に言えば周期が勝手に書き換えられてしまうのです．

１つ試してみましょう．

```
10 PLAY "T05S0M3000L8AA16G16FAG4O6C4","TVLFDEG"
20 PLAY "O5EF16E16DDCE16G16O6C4","EG8E8C4"
```

いかがですか？　S0M3000 にしては極端にエンベロープの周期が短かいですね．PLAY 命令をちょっといじったことのある方ならこのような症状に出合ったことがあるでしょう．これを正常にするには，次のようにエンベロープに関係のないチャネルにもエンベロープ周期(M)の指定をします．

```
10 PLAY "T05S0M3000L8AA16G16FAG4O6C4","TVM3000L
   FDEG"
20 PLAY "O5EF16E16DDCE16G16O6C4","EG8E8C4"
```

今度はまともな音が出ましたね．覚えておくと便利でしょう．

5-3-4　サウンド機能と機械語

PC-6001のサウンド機能は当然機械語で制御できます．その方法について説明しましょう．

サウンド用 LSI とのコミュニケーションのための I／O ポートは，A0～A3H です．（付録-1参照）

このうち A0H がサウンドレジスタのアドレスラッチです．ここにサウンドレジスタの数値を出力することにより，そのレジスタとのコミュニケーションが可能になります．

A1H にデータを出力すると，そのデータが A0H により指定したレジスタに入力され，また A2H には，A0H によって指定したレジスタに入っているデータが現われます．

試しに BASIC の OUT，INP 命令で直接 I／O ポートをアクセスして，SOUND 命令と比べてみましょう．まず，

```
PLAY "a"
```

を実行して下さい．その後で，

```
SOUND 8,8
```

を実行すると，a の音が連続して出ますね．

その後，

```
SOUND 8,0
```

を実行すれば音が止まります．次に続けてＩ／Ｏポートに直接 OUT 命令で出力してみましょう．

```
OUT&HA0,8 : OUT&HA1,8
```

これは SOUND 8,8と同じです．

```
OUT&HA0,8 : OUT&HA1,0
```

これは SOUND 8,0と同じです．

今度はレジスタのデータを読んでみましょう．

```
OUT&HA0,0 : ?  INP(&HA2)
28
```

サウンドレジスタの0番には，28というデータが入っていることが分かりました．

これらはもちろん機械語でも実行できます．

以上のように直接I／Oポートをアクセスすれば，サウンドレジスタにデータが書けることが分かりました．しかしこれはあまりスマートな方法ではありません．そこで，次に N_{60}-BASIC の内部ルーチンを使って，サウンドレジスタを制御してみましょう．

N_{60}-BASIC のサウンドレジスタコントロール用の内部ルーチンは 1BC5H から始まっています．Aレジスタにレジスタ番号，Eレジスタにデータをセットして，このルーチンを CALL すると SOUND 命令と同じ効果が得られます．

たとえば SOUND 7,&HF0 でしたら，

```
0000 3E07    SOUND: LD   A,7
0002 1EF0           LD   E,0F0H
0004 CDC51B         CALL 1BC5H
0007 C9             RET
```

でよいわけです．

ゲームの効果音等，変化の激しい音は BASIC では難しいのですが，このような方法を使って機械語で出せば大変簡単です．

さて，ゲーム等，機械語でつくったプログラム中で困るのは音楽の演奏です．効果音と違って，音楽は音の長さ等を正確に設定してやらねばなりません．そこで，音楽を機械語で演奏するときは PLAY 命令の内部ルーチンを使うのがよいでしょう．

PLAY 命令の内部ルーチンの CALL 番地は 1EB3H です．HL レジスタに，演奏したい音楽を，MML で表記したデータ(BASICのPLAY 命令のときと同じ形式)が格納されているメモリの先頭アドレスを入れて，このルーチンを CALL すれば PLAY 命令と同じ効果が得られます．

次に簡単な例を示します．

```
                      ORG  0D000H
D000 2109D0    PLAY:  LD   HL,DATA
D003 7E               LD   A,(HL)
D004 B7               OR   A
D005 CDB31E           CALL 1EB3H
D008 C9               RET
              ;
D009 2243222C DATA:   DB   '"C","E","G"',0
D00D 2245222C
D011 22472200
```

注） データの終りを示すエンドマーク
の00を忘れないようにすること

前にも述べましたとおり，機械語を使ってサウンド用LSIをコントロールすると，非常に変化の速い複雑な波形をもった音も，簡単に出すことが可能です．

極端な例としては，任意の波形を持った音(たとえば人間の音声)をつくり出すことも可能です．

PC-6001 に使われているサウンド用 LSI は，周波数のレジスタ(たとえばレジスタ 0 とレジスタ 1)に 0 を入れると，出力には音量レジスタ(レジスタ 8 ～10)で指定した直流電圧が発生します(この LSI のマニュアルにはそう発表されてはいないらしいのですが，実際は，こう考えて問題ありません)．ですから，周波数レジスタに 0 を入れておき，音量レジスタの値をすばやく変化させてやれば，任意の波形が出力されるわけです．

最もこれは BASIC では当然速度が不足しますので，機械語でなくてはまともな音は出ません．また音量レジスタの値と実際に出力される電圧は指数関係にあるので，注意してください．

それでは最後にサウンド開発のツールとなるプログラムを公開します．

SOUND 命令で音を上手につくるためのヒケツは，やはり習うより慣れろ，理論よりもまず実践です．自分で PC-6001 を使いながら体得するほかありません．その場合に役に立つのがこのプログラムです．

すべて BASIC で書かれていて，RUN させると画面右側に，reg. NO.？　と聞いてきますのでコントロールしたいレジスタの番号を入力してください．

するとさらに，data？　と聞いてきますのでそのレジスタに入れたいデータを入力してください．画面の左側にはその時点での全サウンドレジスタの内容が表示されます．

なお，レジスタ番号，データはすべて10進数です．入力時に16進数を使いたいときには &HXX という形式で入力してください．

F·1 のキーに&H が入っています．

```
  1 REM**************************
  2 REM*   Sound Creating Tool   *
  3 REM*           for PC-6001   *
  4 REM*    Copyright (C) 1981   *
  5 REM*       Yellow Panther    *
  6 REM**************************
 10 SCREEN 1,1,1:CONSOLE 0,16,0,1:COLOR 0,0,1:CL
    S:KEY1,"&H"
 15 FOR N=0 TO 13:GOSUB 200:NEXT
 20 LOCATE 0,0:PRINT "Reg.--Data":PRINT "-------
    ---"
 30 LOCATE 18,10:PRINT "                ":LOCATE 18,
    10:INPUT"reg.No. ";N
 40 IF N>13 OR N<0 THEN 30
 50 LOCATE 18,10:PRINT "                ":LOCATE 18,
    10:INPUT"    data";D
 60 IF D>255OR N<0 THEN 50
 70 SOUND N,D
100 GOSUB 200
120 GOTO 30
200 LOCATE 0,N+2:OUT &HA0,N:A2=INP(&HA2)
210 PRINT TAB(1+(N>9))N;TAB(8+(A2>9)+(A2>99))A2;
220 RETURN
```

第6章　カセット

第6章　カセット

6-1　ボーレート

PC-6001では，カセットのボーレートを600または1200ボーに切り換えることができます．電源 ON 時では1200ボーにセットされています．

FA1FH 番地がボーレートの切り換えのフラグになっており，電源 ON のときの FA1FH 番地を調べてみましょう．

```
How Many Pages? 2
N60-BASIC
By Microsoft (c) 1981
23484 Bytes free
Ok
?peek(&hfa1f)
 255
Ok
```

FA1FH 番地の値が0の時に600ボー，0以外のときは1200ボーになります．

初期に出荷された製品の取り扱い説明書では，この部分が間違って書かれていますので注意が必要です．

ハード的には，PC-8001，および PC-8801 のカセットを PC-6001 で読み取ることが可能です．

6-2 フォーマット

6-2-1 プログラムファイル

PC-6001では，BASIC のプログラムファイルが，実際どのような型式で SAVE されるか調べてみましょう.

```
100 POKE&HFFE6,0:POKE &HFFE7,&HE
2
110 FOR I=&HFFE8 TO &HFFFB:READ
A$:POKE I,VAL("&h"+A$):NEXT
120 POKE &HFA08,&HE8:POKE &HFA09
,&HFF
200 DATA F5,CD,78,0E,32,1D,FA,E5
,2A,E6,FF,77,23,22,E6,FF,E1,C3,A
6,0F
Ok
RUN
Ok
csave"test"
```

上記のプログラムを実行して，カセットに SAVE します. その後，テープを巻き戻して，今，SAVE したプログラムを LOAD します.

```
cload
Found:test
Ok
```

LOAD 終了後[ページ]キーを押してページ 2 に切り換えます.

```
ƐƐƐƐƐƐƐƐƐƐtest  太Od え&HFFE6,0:え
&HFFE7,&HE2 KOn ♥ I×&HFFE8 テ &HF
FFB:を A$:え I,め("&h"ハA$):♣ kO× え
&HFA08,&HE8:え &HFA09,&HFF ュOネ ◆
F5,CD,78,0E,32,1D,FA,E5,2A,E6,FF
,77,23,22,E6,FF,E1,C3,A6,0F
```

ページ 2 に表示されている内容が，テープに記録されていた内容になります.

	ヘッダ										ファイル名					
E200	D3	D3	D3	D3	D3	D3	D3	D3	D3	D3	74	65	73	74	00	00
E210	1D	84	64	00	94	26	48	46	46	45	36	2C	30	3A	94	20
E220	26	48	46	46	45	37	2C	26	48	45	32	00	4B	84	6E	00
E230	81	20	49	D2	26	48	46	46	45	38	20	C3	20	26	48	46

図6-1　プログラムファイルのフォーマット

図6-1を見てください．まずモーターをONにして3.4秒間テープをから送りします．(これはモーターの回転を安定させるために必要です)

次に同じく3.4秒間，マーク(2400Hz 4サイクル)を出力します．そしてヘッダ・マークであるD3Hを10個書き込みます．LOADするときにD3Hを見て，プログラム・ファイルであると判断されます．

次にファイル名を6バイト書くわけですが，もしファイル名が6文字に満たないときは，後に00を書き込みます．

続いて0.05秒間テープを進めます．これはこのときに，LOAD時のファイル名チェックや，Found： … ，Skip 表示などの処理を行なわせるためです．

この後よりプログラムになります．(FA5F, 60H)で指されるプログラムの先頭番地から(FF56, 57Hの変数開始番地)-1までをカセットに出力します．これはメモリに入っている内容がそのまま送られます．

最後にエンドマークの00を9つ書き込みます．プログラムの最後に00を3つ送っていますので00は合計12個書き込まれることになります．

LOADのときに00が10個連続していればファイルエンドとみなされます．

データをすべて転送した後はモーターをOFFにしてSAVEを終わります．

6-2-2　データ・ファイル

データ・ファイルの構造は次のようになっています．

図6－2　データ・ファイルのフォーマット

6.8秒の待ち時間の後9CHを6個ヘッダに書き込みます．データの型式は文字列はそのまま書き込まれ，数値はASCIIコードに変換されて書き込まれます．データエンドとして0DH(CR)を書き込んでデータ終了になります．

6-3　PC-8001のデータをPC-6001で使用する

PC-6001とPC-8001とでは，データ・フォーマットが同じですのでボーレートさえ合わせればPC-8001のテープをPC-6001で読むことができます．本当に読めるかどうかテストしてみましょう．

PC-8001をお持ちの方は次のプログラムを実行してみて下さい．

```
100 CLEAR 1000:DIM DA(10),DA$(10)
110 FOR I=1 TO 10:READ DA(I):NEXT I
120 FOR I=1 TO 10:READ DA$(I):NEXT I
140 FOR I=1 TO 10:PRINT #-1,DA(I),DA$(I):NEXT I
200 DATA 1,2,3,4.5,5.3,6.4,7.7,8,9,10
210 DATA AB,BC,CD,DE,EF,FG,GH,HI,IJ,JK
```

※注　これはPC-8001のプログラムです。

このプログラムでつくられたデータ・テープをPC-6001で読み取ってみます．

```
100 CLEAR 1000:DIM DA(10),DA$(10
)
110 POKE &HFA1F,0
140 FOR I=1 TO 10:INPUT #-1,DA(I
),DA$(I):NEXT I
150 FOR I=1 TO 10:PRINT DA(I),DA
$(I):NEXT I
Ok
```

（110行の注）ボーレートを600ボーにセットする。

```
RUN
 1            AB
 2            BC
 3            CD
 4.5          DE
 5.3          EF
 6.4          FG
 7.7          GH
 8            HI
 9            IJ
 10           JK
Ok
```

```
1 COLOR  CLOAD  GOTO  LIST  RUN
```

PC-6001で上記のプログラムを実行すると，PC-8001 で SAVE されたデータ・ファイルが正確に読み取られています.

同様に PC-8801 のデータ・テープも読み取ることができます.

```
100 CLEAR 1000:DIM DA(10),DA$(10)
110 FOR I=1 TO 10:READ DA(I):NEXT I
120 FOR I=1 TO 10:READ DA$(I):NEXT I
130 OPEN "CAS1:data" FOR OUTPUT AS #1
140 FOR I=1 TO 10:PRINT #1,DA(I),DA$(I):NEXT I
200 DATA 1,2,3,4.5,5.3,6.4,7.7,8,9,10
210 DATA AB,BC,CD,DE,EF,FG,GH,HI,IJ,JK
```

※PC-8801 のプログラム

```
100 CLEAR 1000:DIM DA(10),DA$(10)
110 POKE &HFA1F,1 ←———————— 1200ボーにセット
140 FOR I=1 TO 10:INPUT #-1,DA(I),DA$(I):NEXT I
150 FOR I=1 TO 10:PRINT DA(I),DA$(I):NEXT I
```

6-4 PC-8001のプログラムテープをPC-6001でLOADする

PC-8001 のデータ・テープを PC-6001 で読むことができるならば，プログラムテープもハード的には読むことができるはずです．

先ほどのプログラムを SAVE して PC-6001 で LOAD してみましょう．

```
100 CLEAR 1000:DIM DA(10),DA$(10)
110 FOR I=1 TO 10:READ DA(I):NEXT I
120 FOR I=1 TO 10:READ DA$(I):NEXT I
140 FOR I=1 TO 10:PRINT #-1,DA(I),DA$(I):NEXT I
200 DATA 1,2,3,4.5,5.3,6.4,7.7,8,9,10
210 DATA AB,BC,CD,DE,EF,FG,GH,HI,IJ,JK
```

⇓

※PC-8001のプログラム

```
LIST
100 LPRINT  ASC:READ DA(
                        ),DA$(
                             )
110 NEXT ITIME USR
                  :LET DA(I):DA
TA I
120 NEXT ITIME USR
                  :LET DA$(I):D
ATA I
140 NEXT ITIME USR
                  :ON #47:8 ,DA
(I),DA$(I):DATA I
200 INPUT 1,2,3,4.5,5.3,6.4,7.7,
8,9,10
210 INPUT AB,BC,CD,DE,EF,FG,GH,H
I,IJ,JK
Ok
```

※PC-6001でLOADし，LISTしたときのプログラム，
PC-6001のボーレートは600ボーにすること

PC-8001 と PC-6001 では，中間言語や数値型の格納の仕方の違いから LIST を取っても正常なプログラムになっていません．

プログラムを中間言語レベルではなく，文字列の型でSAVEを行なえば，少なくとも PC-6001 で命令が変わることはありません．

プログラムを文字列でSAVEする一番簡単な方法は，プログラムの先頭に，" をつければよいのです．

こうすると中間言語交換のときに，" があるために文字列と解釈されて中間言語に変換されません．

```
100 "CLEAR 1000:DIM DA(10),DA$(10)
110 "FOR I=1 TO 10:READ DA(I):NEXT I
120 "FOR I=1 TO 10:READ DA$(I):NEXT I
140 "FOR I=1 TO 10:PRINT #-1,DA(I),DA$(I):NEXT I
200 "DATA 1,2,3,4.5,5.3,6.4,7.7,8.9,10
210 "DATA AB,BC,CD,DE,EF,FG,GH,HI,IJ,JK
```

⇓ 各ステートメントの先頭に "をつけて，PC-8001で SAVE します，そして PC-6001で LOAD します

```
100 "CLEAR 1000:DIM DA(10),DA$(1
0)
110 "FOR I=1 TO 10:READ DA(I):NE
XT I
120 "FOR I=1 TO 10:READ DA$(I):N
EXT I
140 "FOR I=1 TO 10:PRINT #-1,DA(
I),DA$(I):NEXT I
200 "DATA 1,2,3,4.5,5.3,6.4,7.7,
8.9,10
210 "DATA AB,BC,CD,DE,EF,FG,GH,H
I,IJ,JK
```

⇓ "を取り除きます

```
100 CLEAR 1000:DIM DA(10),DA$(10)
110 FOR I=1 TO 10:READ DA(I):NEXT I
120 FOR I=1 TO 10:READ DA$(I):NEXT I
140 FOR I=1 TO 10:PRINT #-1,DA(I),DA$(I):NEXT I
200 DATA 1,2,3,4.5,5.3,6.4,7.7,8.9,10
210 DATA AB,BC,CD,DE,EF,FG,GH,HI,IJ,JK
```

この程度のプログラムでしたら PC-6001 でも実行できますが，PC-6001 にない命令を使用していた場合はプログラムの修正が必要になります．

6-5　BASICと機械語を一度にSAVE・LOADする

SAVE のときに，FA5F, 60H で指される番地から，FF56, 57Hで指されるメモリのアドレス-1までテープに書き込まれます．そしてLOADするときは00が10個でてくるまで読みます．このことはSAVE するときに FF56, 57H の番地の値を変更すれば BASIC とその後に続く機械語のプログラムを SAVE できることになります．

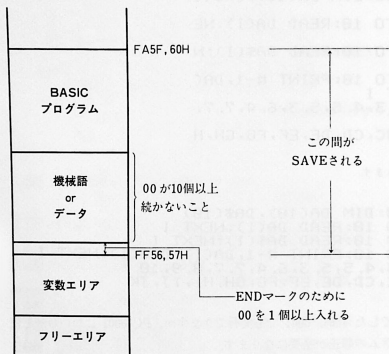



図6-3

これを行なう方法は次のとおりです．

1. BASIC のプログラムを入力する．
2. FF56, 57H の値を書き換えて機械語の入るエリアを確保する．
3. 機械語のプログラムを入力する．(機械語のプログラムの最後には必ず00が1個以上あること)
4. CSAVE を行なう．

ただし，注意することが3つあります．

1. FF56, 57H のポインタを書き換えたあとはBASIC のプログラムを修正してはいけません．もし修正を行なうと，プログラムが増減した分だけ機械語の位置がずれます．
2. 機械語のプログラムの途中に00が10個以上あってはいけません．
3. 機械語のプログラムをリロケータブルにつくっていない限り，SAVE したときのRAM容量と同じでないと LOAD したときに正常に動きません．

市販のソフトには，このようにBASIC＋機械語の形でSAVEされているプログラムを多数みかけます．

6-6 データファイルにおける，と；との違い

複数のデータを1度にカセットにSAVEしようとするとき，PC-8001では，PRINT#-1，A，B，C，Dのように行なえばよかったのですが，PC-6001では，PRINT#-1，A，","，B，","，C，","，Dのようにしませんと INPUT#-1で読み込んだときにFDエラーになります．PC-8001の場合はデータの区切りとして","を自動的に送り出すのに対して，PC-6001は送り出さないために変数AとBとがつながって文字列とみなされるために FD Error(File Data Error)となるのです．

そのため PC-6001 はセパレータとしてプログラムで，","を書く必要があります．

ところで PC-6001 ではPRINT#-1，A，","，Bとするよりも PRINT#-1，A;",";Bとする方が，データの SAVE の時間が短くてすみます．これはなぜでしょうか？

まず，','を使った場合を調べてみましょう．

```
 10 FOR I=1 TO 5:READ A$:A=VAL("&H"+A$)
 20 PRINT #-1,A$,",",A
 30 NEXT
 40 PRINT "テープ ヲ マキモドシテ クダサイ"
 50 PRINT "OK ナラ RETURN KEY ヲ オシテ クダサイ"
 60 INPUT A$
 70 RESTORE 200
100 POKE&HFFE6,0:POKE&HFFE7,&HE2
110 FOR I=&HFFE8 TO &HFFFB:READ A$:POKE I,VAL("&
    h"+A$):NEXT
120 POKE &HFA08,&HE8:POKE &HFA09,&HFF
150 FOR I=1 TO 5:INPUT #-1,A$,A:PRINT A$;A:NEXT
200 DATA F5,CD,78,0E,32,1D,FA,E5,2A,E6,FF,77,23,
    22,E6,FF,E1,C3,A6,0F
```

テープを録音状態にしてプログラムを実行します．リレーがカッチ，カッチという音をたててデータがSAVEされていきます．そしてしばらくすると，〝テープヲマキモドシテクダサイ″のメッセージが表示されますのでテープを巻き戻し，再生状態にして RETURN キーを押します．そうしますとテープが回りだしデータを読み込みます．

```
RUN
テープ ヲ マキモドシテ クダサイ
OK ナラ RETURN KEY ヲ オシテ クダサイ
?
F5          245
CD          205
78          120
0E          14
32          50
Ok
```

ここで[,]キーを押してページ2に切り換えると今のテープフォーマットが表示されます．

```
LLLLLLLF5                   ,
          245百LLLLLLLCD
,                                205百LLLLLLL78
                                          120百LL
                  ,
LLLLL0E                   ,
     14百LLLLLLL32                 ,
                   50百
```

ページ2の表示

","のためにスペースが14個出力されている

```
E200 9C 9C 9C 9C 9C 9C 46 35 20 20 20 20 20 20 20 20
E210 20 20 20 20 20 20 2C 20 20 20 20 20 20 20 20 20
E220 20 20 20 20 20 20 32 34 35 0D 9C 9C 9C 9C 9C 9C
E230 43 44 20 20 20 20 20 20 20 20 20 20 20 20 20 20
E240 2C 20 20 20 20 20 20 20 20 20 20 20 20 20 20 20
E250 32 30 35 0D 9C 9C 9C 9C 9C 9C 37 38 20 20 20 20
E260 20 20 20 20 20 20 20 20 20 20 2C 20 20 20 20 20
E270 20 20 20 20 20 20 20 20 20 20 31 32 30 0D 9C 9C
E280 9C 9C 9C 9C 30 45 20 20 20 20 20 20 20 20 20 20
E290 20 20 20 20 2C 20 20 20 20 20 20 20 20 20 20 20
E2A0 20 20 20 20 31 34 0D 9C 9C 9C 9C 9C 9C 33 32 20
E2B0 20 20 20 20 20 20 20 20 20 20 20 20 20 2C 20 20
E2C0 20 20 20 20 20 20 20 20 20 20 20 20 20 35 30 0D
```

次に20行を

20 PRINT#-1,A$;",";A

に変更して同じように実行してみます．

```
RUN
テープ ヲ マキモドシテ クダサイ
OK ナラ RETURN KEY ヲ オシテ クダサイ
?
F5 245
CD 205
78 120
0E 14
32 50
Ok
```

```
ᄔᄔᄔF5, 245ᆷᄔᄔᄔCD, 205ᆷᄔᄔ
ᄔ78, 120ᆷᄔᄔᄔ0E, 14ᆷᄔᄔᄔ32,
 50ᆷ
```

ページ 2 の表示

```
E200 9C 9C 9C 9C 9C 9C 46 35 2C 20 32 34 35 0D 9C 9C
E210 9C 9C 9C 9C 43 44 2C 20 32 30 35 0D 9C 9C 9C 9C
E220 9C 9C 37 38 2C 20 31 32 30 0D 9C 9C 9C 9C 9C 9C
E230 30 45 2C 20 31 34 0D 9C 9C 9C 9C 9C 9C 33 32 2C
E240 20 35 30 0D 20 20 20 20
```

実行結果から分かるようにカンマ(,)を使うよりもセミコロン(;)の方がデータの長さが短くなります．カンマを使用した場合は 2 バイトの文字列データだったのがスペースが付加されて16バイトになっています．

ところで，前述の PRINT#-1 文中の"；"は省略可能です．先ほどのプログラムの20行を，

```
20 PRINT#-1, A$",''A
```

と変更して実行してみてください．"；"を入れたときと同じ結果になるでしょう．

PRINT#-1 を使うときに問題になるのは長いデータは SAVE できないということです．

```
  5 CLEAR 300
 10 A$="0123456789":DA$=A$+A$+A$+A$+A$+A$+A$+A$+
    A$+A$:A$=DA$+DA$
 20 PRINT #-1,A$
 40 PRINT "テープ ヲ マキモドシテ クダサイ!
 50 PRINT "OK ナラ return key ヲ オシテクダサイ"
 60 INPUT A$
 70 RESTORE 200
100 POKE &HFFE6,0:POKE &HFFE7,&HE2
110 FOR I=&HFFE8 TO &HFFFB:READ A$:POKE I,VAL("&
    h"+A$):NEXT
120 POKE &HFA08,&HE8:POKE&HFA09,&HFF
150 INPUT #-1,A$:PRINT A$
200 DATA F5,CD,78,0E,32,1D,FA,E5,2A,E6,FF,77,23,
    22,E6,FF,E1,C3,A6,0F
```

実験的に200バイトの文字列をSAVEして，その後にこれを LOADしてみます．

```
RUN
テープ ヲ マキモドシテ クダサイ!
OK ナラ return key ヲ オシテクダサイ
?
01234567890123456789012345678901
23456789012345678901234567890123
4567890
Ok
```

```
LLLLLL012345678901234567890123456789012345
678901234567890123456789012345678901234567
89012345678901
```

ページ２の画面

これで分かるように１つのデータブロックは最大71文字しか使えません．これはバッファが72文字分しかないために起こります．もし長い文字列をSAVEする場合は文字列を分解してSAVEするなどの工夫がいります．

第７章　プリンタ出力

第7章　プリンタ出力

7-1　PC-6021

PC-6021は感熱式のプリンタで，PR-1001(PC-8022)と比べて，ビデオコピーの機能を取り除いている他は，機能的に変わるところはありません(もしPC-8022をお持ちの方はPC-6001で使用できます)．

このプリンタは6つのコマンドを備えています．

機　能	略　号	コード(16進)	使い方
紙送り(0～255行)	F／F	0C＋データ	CHR$(12)＋CHR$(n) n＝紙送りの行数
右づめ印字	RA	12	CHR$(18)
白黒反転 (グラフィックモード時)	REVERSE	14	※文字の白黒反転は BASICではできない
グラフィックプリント	GS	1D＋データ	CHR$(29)＋CHR$(L) ＋CHR$(データ)… CHR$(データ) L＝グラフィック印字をするライン数
モードのリセット	RS	1E	CHR$(30)
拡大印字	US	1F	CHR$(31)

注：14HのみCHR$命令で送ることができません．白黒反転を行う場合は機械語を使用します．

例
```
LD A, 14H
CALL    1A16H;  プリンタ1文字出力
RET
```

図7-1　プリンタ・コマンド一覧表

7-2 キャラクタ

PC-6021が印字できるキャラクタは，英記号，英文字，カナに限られており，PC-6001 内部で持っているグラフィック，漢字等はプリンタに出力することができません．

PC-6021 キャラクタ表

```
   !  "  #  $  %  &  '  (  )  *  +  ,  -  .  /
0  1  2  3  4  5  6  7  8  9  :  ;  <  =  >  ?
@  A  B  C  D  E  F  G  H  I  J  K  L  M  N  O
P  Q  R  S  T  U  V  W  X  Y  Z  [  ¥  ]  ^  _
`  a  b  c  d  e  f  g  h  i  j  k  l  m  n  o
p  q  r  s  t  u  v  w  x  y  z  {  ¦  }  ~
         ヲ ァ ィ ゥ ェ ォ ャ ュ ョ ッ
ー ア イ ウ エ オ カ キ ク ケ コ サ シ ス セ ソ
タ チ ツ テ ト ナ ニ ヌ ネ ノ ハ ヒ フ ヘ ホ マ
ミ ム メ モ ヤ ユ ヨ ラ リ ル レ ロ ワ ン ゛ ゜
```

ただし，ひらがなだけはカタカナに変換して出力されます．

例　LPRINT〝あいうえお〞

　　アイウエオ

PC-6001 には，(N_{60}-BASIC が特殊記号を出力しないように制限しているために，)セントロニクス仕様のプリンタであればどの機種でも接続することができます．しかし，画面 COPYのコントロール(LCOPY)が PC-6021 の機能仕様になっているため，PC-6001専用となっているプリンタ以外は，基本的には LCOPY 命令は使えません．

N_{60}-BASIC がプリントアウトすることのできるキャラクタを下記に示します．

キャラクタ表示プログラム

```
10 FOR  I=32 TO 255 STEP 4:FOR J=0TO 3
20 LPRINT TAB(8*J);I+J;"=";CHR$(34);CHR$(I+J);C
   HR$(34);
30 NEXT J:LPRINT:NEXT I
```

```
32=" "   33="!"   34="""   35="#"
36="$"   37="%"   38="&"   39="'"
40="("   41=")"   42="*"   43="+"
44=","   45="-"   46="."   47="/"
48="0"   49="1"   50="2"   51="3"
52="4"   53="5"   54="6"   55="7"
56="8"   57="9"   58=":"   59=";"
60="<"   61="="   62=">"   63="?"
64="@"   65="A"   66="B"   67="C"
68="D"   69="E"   70="F"   71="G"
```

```
72="H"   73="I"   74="J"   75="K"
76="L"   77="M"   78="N"   79="O"
80="P"   81="Q"   82="R"   83="S"
84="T"   85="U"   86="V"   87="W"
88="X"   89="Y"   90="Z"   91="["
92="¥"   93="]"   94="^"   95="_"
96="`"   97="a"   98="b"   99="c"
100="d"  101="e"  102="f"  103="g"
104="h"  105="i"  106="j"  107="k"
108="l"  109="m"  110="n"  111="o"
112="p"  113="q"  114="r"  115="s"
116="t"  117="u"  118="v"  119="w"
120="x"  121="y"  122="z"  123="{"
124="|"  125="}"  126="~"  127=" "
128="" 129="" 130="" 131=""
132="" 133="" 134="ｦ" 135="ｧ"
136="ｨ"  137="ｩ"  138="ｪ"  139="ｫ"
140="ｬ"  141="ｭ"  142="ｮ"  143="ｯ"
144="ｰ"  145="ｱ"  146="ｲ"  147="ｳ"
148="ｴ"  149="ｵ"  150="ｶ"  151="ｷ"
152="ｸ"  153="ｹ"  154="ｺ"  155="ｻ"
156="ｼ"  157="ｽ"  158="ｾ"  159="ｿ"
160=" "  161="｡"  162="｢"  163="｣"
164="､"  165="･"  166="ｦ"  167="ｧ"
168="ｨ"  169="ｩ"  170="ｪ"  171="ｫ"
172="ｬ"  173="ｭ"  174="ｮ"  175="ｯ"
176="ｰ"  177="ｱ"  178="ｲ"  179="ｳ"
180="ｴ"  181="ｵ"  182="ｶ"  183="ｷ"
184="ｸ"  185="ｹ"  186="ｺ"  187="ｻ"
188="ｼ"  189="ｽ"  190="ｾ"  191="ｿ"
192="ﾀ"  193="ﾁ"  194="ﾂ"  195="ﾃ"
196="ﾄ"  197="ﾅ"  198="ﾆ"  199="ﾇ"
200="ﾈ"  201="ﾉ"  202="ﾊ"  203="ﾋ"
204="ﾌ"  205="ﾍ"  206="ﾎ"  207="ﾏ"
208="ﾐ"  209="ﾑ"  210="ﾒ"  211="ﾓ"
212="ﾔ"  213="ﾕ"  214="ﾖ"  215="ﾗ"
216="ﾘ"  217="ﾙ"  218="ﾚ"  219="ﾛ"
220="ﾜ"  221="ﾝ"  222="ﾞ"  223="ﾟ"
224="ﾀ"  225="ﾁ"  226="ﾂ"  227="ﾃ"
228="ﾄ"  229="ﾅ"  230="ﾆ"  231="ﾇ"
232="ﾈ"  233="ﾉ"  234="ﾊ"  235="ﾋ"
236="ﾌ"  237="ﾍ"  238="ﾎ"  239="ﾏ"
240="ﾐ"  241="ﾑ"  242="ﾒ"  243="ﾓ"
244="ﾔ"  245="ﾕ"  246="ﾖ"  247="ﾗ"
248="ﾘ"  249="ﾙ"  250="ﾚ"  251="ﾛ"
252="ﾜ"  253="ﾝ"  254="ﾞ"  255="ﾟ"
```

7-3 画面コピーの方法

PC-8001 の場合は，画面コピーをとるには市販の画面コピー用の ROM を購入しなければなりませんでしたが，PC-6001 では LCOPY 命令で簡単に行なえます．

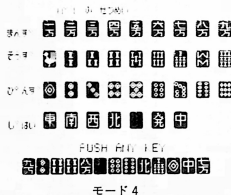
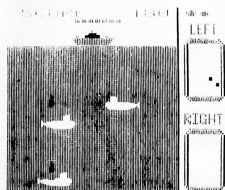
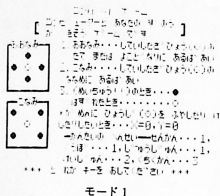
7-3-1 PC-6021による画面コピー

画面コピーの例

モード 1

モード 2

モード 3

モード 4

画面コピーは256×192のドットで印字されており，このドット数はちょうど，画面のドット数と同じになっています．そのため画面コピーされたものは，カラーを濃淡で表現することができません．

一見，モード 3 は濃淡で表われているように見えますが，プログラム的には，カラー対応になっていません．VDG のグラフィックモードを考えれば，この理由が簡単に分かるはずです．

LCOPY のプログラムで，画面モード 3，4 では単にビットイメージのデータとして VRAM の内容をプリンタに転送しているので，128×192モードでは 2 ドットで 1 つの点の色を表わしていましたから，その色によってドットのデータが変わるために濃淡として見えるだけなのです．

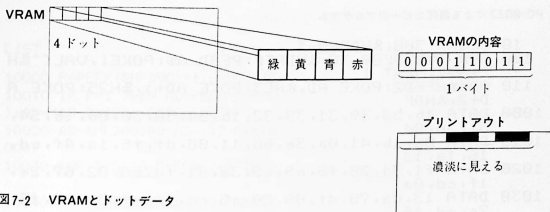



図7-2　VRAMとドットデータ

7-3-2　他のプリンタによる画面コピー

ビットイメージの機能を持つプリンタならば，画面コピーのプログラムをつくることによりコピーが可能になります．

N_{60}-BASIC では，他のプリンタで画面コピーができるように考慮されています．

LCOPY 命令があると，まずFFD2H 番地をコールするようになっています．そこでこの番地を画面コピーのプログラムのアドレスに書き換えて，スタックポインタを細工すれば自分のプリンタにあった LCOPY が可能となります．

例として PC-8023 で画面コピーを行なうプログラムを示します．

画面コピーの例

モード1　モード2　モード3　モード4

PC-8023による画面コピーのプログラム

```
  10 CLEAR 300,&HDF00-1
 100 FOR I=&HDF00 TO &HDFFF:READ A$:POKEI,VAL("&H
     "+A$):NEXT I
 110 AD=&HFFD2:POKE AD,&HC3:POKE AD+1,&H25:POKE A
     D+2,&HDF
1000 DATA 1b,53,30,31,39,32,1b,54,30,30,0d,1b,54,
     31,36,0d
1010 DATA 0a,1b,41,0a,3e,06,11,00,df,f5,1a,4f,cd,
     16,1a,13
1020 DATA f1,3d,20,f5,c9,e5,3a,91,fd,c6,02,67,2e,
     1f,cd,8a
1030 DATA 13,da,70,df,06,20,e5,cd,14,df,3e,c0,f5,
     7e,cd,16
1040 DATA 1a,11,20,00,19,f1,3d,20,f3,3e,0b,11,06,
     df,cd,19
1050 DATA df,e1,2b,10,e1,3e,03,11,11,df,cd,19,df,
     3e,0a,06
1060 DATA 06,cd,1c,1a,10,fb,e1,d1,c9,00,00,00,00,
     00,00,00
1070 DATA 06,20,e5,cd,14,df,3e,10,f5,e5,25,25,7e,
     e6,40,eb
1080 DATA e1,e5,c5,0e,00,cc,a4,df,c4,c9,df,c1,e1,
     11,20,00
1090 DATA 19,f1,3d,20,e3,3e,0b,11,06,df,cd,19,df,
     e1,2b,10
1100 DATA d1,c3,55,df,e5,7e,d5,cd,a0,14,eb,06,00,
     09,3e,04
1110 DATA d3,93,d1,1a,0f,7e,30,01,2f,cd,16,1a,e1,
     0c,79,fe
1120 DATA 0c,20,e1,3e,05,d3,93,af,c9,e5,c5,79,0f,
     0f,e6,03
1130 DATA 2f,c6,04,4f,7e,0d,28,04,0f,0f,18,f9,e6,
     03,4f,06
1140 DATA 00,21,4f,23,09,7e,cd,16,1a,c1,e1,0c,79,
     fe,0c,20
1150 DATA d8,c9,00,00,00,00,00,00,00,00,00,00,00,
     00,00,00
```

7-3-3　キャラクタのみ画面コピーする方法

キャラクタコピーだけでよければ，BASICで簡単につくることができます．

キャラクタ・コピー　プログラム

```
10000 P=PEEK(&HFD90)+1:S=0:E=15
10010 IF P=1 THEN AD=PEEK(&HFA60):AD=(AD-2)*256:GO
      TO 10030
10020 AD=&HE200:AD=AD+((2-P)*&H2000)
10030 FOR I=S TO E:FOR J=0 TO 31
10040 DA=PEEK(AD+32*I+J):LPRINT CHR$(DA);:NEXT J:L
      PRINT :NEXT I
10050 RETURN
```

実行例

```
LIST

10000 P=PEEK(&HFD90)+1:S=0:E=15
10010 IF P=1 THEN AD=PEEK(&HFA60
):AD=(AD-2)*256:GOTO 10030
10020 AD=&HE200:AD=AD+((2-P)*&H2
000)
10030 FOR I=S TO E:FOR J=0 TO 31

10040 DA=PEEK(AD+32*I+J):LPRINT
CHR$(DA);:NEXT J:LPRINT :NEXT I
10050 RETURN
Ok
gosub 10000

1 COLOR CLOAD GOTO  LIST  RUN
```

このプログラムはサブルーチン形式になっています．Pにコピーしたいページ，Sにプリント開始行，Eに最終行を入れて GOSUB 10010 でコールすれば，そのページのSとEで指定した範囲のコピーを行ないます．

7-4 PC-8023によるひらがな出力

PC-8023 には，〝ひらがな〟を印示する機能がありますので，これを使ってひらがなを出力させてみましょう．

ひらがな出力プログラム

```
  10 REM PC-8023 ﾋﾗｶﾞﾅ
  20 CLEAR 300,&HDF00-1
 100 FOR I=&HDF00 TO &HDF70:READ A$:POKEI,VAL("&h
     "+A$):NEXT I
 110 POKE&HFFD1,&HDF:POKE&HFFD0,&H0:POKE&HFFCF,&H
     C3
 120 POKE&HFFE4,0
1000 DATA f5,3a,58,fa,fe,01,28,02,f1,c9,f1,33,33,
     f5,fe,09
1010 DATA 20,0e,3e,20,cd,c7,26,3a,57,fa,e6,07,20,
     f4,f1,c9
1020 DATA f1,f5,d6,0d,28,06,38,07,3a,57,fa,3c,32,
     57,fa,f1
```

```
1030 DATA f5,c5,fe,86,38,22,fe,a0,38,04,fe,e0,38,
     1a,ee,20
1040 DATA f5,3a,e4,ff,b7,20,0d,3e,1b,cd,16,1a,3e,
     26,cd,16
1050 DATA 1a,32,e4,ff,f1,c3,29,1a,f5,3a,e4,ff,b7,
     28,0e,3e
1060 DATA 1b,cd,16,1a,3e,24,cd,16,1a,af,32,e4,ff,
     f1,c3,29
1070 DATA 1a
```

```
 10 REM あいうえお
 20 REM かきくけこ
 30 REM さしすせそ
 40 REM たちつてと
 50 REM なにぬねの
 60 REM はひふへほ
```

このプログラムは DF00～DF70H までを使っています．ページ数の指定によってはリロケートの必要があります．

7-5 PC-8023によるグラフィック出力

つぎに，PC-8023 にグラフィックキャラクタを，出力させてみましょう．

PC-6001 のグラフィックパターンは PC-8023 にはないものが多いため，プリンタに出力しようとすると，どうしてもビットイメージで出力するしか方法がありません．また，キャラクタによっては，コントロールコードと重なる部分があるので，これらの処理が必要となります．プログラム自体は簡単なのですが，ビットイメージのデータでかなりメモリを専有します．

グラフィックパターンは 8 ×12ドットにしなければならないのですが，プログラムの簡素化のために 8 × 8 ドットで行なっていますので，すこし見づらくなります．

なおこのプログラムには，ひらがな出力プログラムも入れておきます．

```
  10 CLEAR 300,&HDDFF
  20 FOR I=&HDE00 TO &HDFCF:READ A$:POKE I,VAL("&
     h"+A$):NEXT
  30 POKE &HFFD1,&HDE:POKE &HFFD0,&H0:POKE &HFFCF
     ,&HC3
1000 DATA f5,3a,58,fa,fe,01,28,02,f1,c9,f1,33,33,
     f5,fe,09
1010 DATA ca,e0,26,d6,0d,ca,fa,26,da,fa,26,3a,57,
     fa,3c,32
1020 DATA 57,fa,f1,e5,c5,cd,8b,10,da,4c,1a,05,28,
     33,fe,86
1030 DATA 38,23,fe,a0,38,04,fe,e0,38,1b,c3,c0,df,
     3e,1b,cd
1040 DATA 16,1a,3e,26,cd,16,1a,f1,cd,16,1a,3e,1b,
     cd,16,1a
```

```
1050 DATA 3e,24,c3,29,1a,fe,80,da,29,1a,fe,86,d2,
     29,1a,d6
1060 DATA 60,e5,d5,26,00,6f,29,29,29,11,90,de,19,
     11,87,de
1070 DATA 06,06,1a,cd,16,1a,13,10,f9,06,08,7e,cd,
     16,1a,23
1080 DATA 10,f9,d1,e1,c1,f1,c9,1b,53,30,30,30,38,
     00,00,00
1085 DATA 00,00,00,00,00,00,00,00,00,80,7f,29,29,
     29,ff,00
1090 DATA 80,46,20,1f,20,46,80,00,44,24,1c,ff,0c,
     32,c1,00
1100 DATA c4,34,0c,ff,0c,34,c4,00,88,cc,aa,f9,aa,
     cc,88,00
1110 DATA 40,40,44,7f,44,40,40,00,00,7f,49,49,49,
     7f,00,00
1120 DATA 24,33,2c,fe,2c,22,20,00,ff,09,09,0f,09,
     89,ff,00
1130 DATA 3e,2a,3e,68,aa,ff,2a,20,08,cc,3a,09,8a,
     7c,08,00
1140 DATA 2c,fe,29,1c,80,7f,28,10,f9,a9,ad,ab,a9,
     f9,00,00
1150 DATA 10,14,14,fe,11,10,10,00,20,19,0d,0b,89,
     f9,00,00
1160 DATA 00,42,3e,02,7e,42,62,00,18,18,18,1f,1f,
     18,18,18
1170 DATA 18,18,18,f8,f8,18,18,18,18,18,18,ff,ff,
     18,18,18
1180 DATA 00,00,00,ff,ff,18,18,18,18,18,18,ff,ff,
     18,18,18
1190 DATA 00,00,00,ff,ff,00,00,00,18,18,18,18,18,
     18,18,18
1200 DATA 00,00,00,f8,f8,18,18,18,18,18,18,f8,f8,
     00,00,00
1210 DATA 00,00,00,1f,1f,18,18,18,18,18,18,1f,1f,
     00,00,00
1220 DATA 00,42,24,18,18,24,42,00,84,44,24,1f,24,
     44,84,84
1230 DATA 0e,0a,0a,ff,0a,0a,0e,00,20,18,80,ff,00,
     08,10,20
1240 DATA 30,9c,de,ff,de,9c,30,00,0e,3f,7e,fc,7e,
     3f,0e,00
1250 DATA 1c,9c,df,ff,df,9c,1c,00,08,1c,3e,7f,3e,
     1c,08,00
1260 DATA 3c,42,81,81,81,81,42,3c,3c,7e,ff,ff,ff,
     ff,7e,3c
1270 DATA fe,fe,30,02,ee,20,f5,c3,3d,de,00,00,00,
     00,00,00
```

第8章　N_{60}－BASICの命令分析

第8章　N_{60}-BASICの命令分析

8-1　N_{60}-BASICとN-BASICの命令比較

N_{60}-BASICおよびN-BASICは同じマイクロソフト社製のBASICですので，基本的な命令は同じになっていますが，ハードや，使用目的の違いから，画面処理やサウンド機能で大きな差異が見られます．

(1)　N_{60}-BASIC，N-BASICとも同一機能

ABS	AND	ASC	CHR$
CLEAR	CLOAD	CLOAD?	CONT
COS	CSAVE	CSRLIN	DATA
DEFFN	DIM	END	EXP
FOR	FRE	GOSUB	GOTO
IF	INKEY$	INP	INPUT
INPUT#	INT	KEY	LEFT$
LEN	LET	LIST	LLIST
LOG	LPOS	LPRINT	MID$
NEXT	NEW	NOT	ON GOSUB
ON GOTO	OR	OUT	PEEK
POKE	PRINT	PRINT#	READ
REM	RESTORE	RETURN	RIGHT$
RND	RUN	SGN	SIN
SPC	SQR	STEP	STOP
STR$	TAB	TAN	THEN
VAL			

(2)　N_{60}-BASICとN-BASICではパラメータが異なるもの

COLOR	CONSOLE	LINE	LOCATE
POINT	PRESET	PSET	USR
TIME			

（3） N_{60}-BASIC ではけずられたもの

ATN	AUTO	BEEP	CDBL
CINT	CSNG	DEFDBL	DEFINT
DEFSNG	DEFSTR	DEFUSR	DELETE
ELSE	EQV	ERASE	ERL
ERR	ERROR	FIX	GET@
HEX$	IMP	INPUT$	INSTR
KEYLIST	LINEINPUT	LPRINTUSING	MOD
MON	MOTOR	OCT$	ON ERROR GOTO
PRINT USING	PUT@	RENUM	SPACE$
SWAP	TERM	TIME$	TROFF
TRON	VARPTR	WAIT	WIDTH
XOR			

（4） N_{60}-BASIC にしかないもの

CLS	EXEC	LCOPY	PAINT
PLAY	SCREEN	STICK	STRIG
SOUND			

（1）に属する部分は BASIC の基本的な命令で使い方に他機種との差異はありません．

（2）の部分はおもにハードウェアの違いによるもので，PC-8001 のグラフィックは160×100ドット，カラー8色が使えましたが，CRT コントローラの LSI の違いから，PC-6001 ではグラフィックモードが3種類選べ，おのおのモードで使用できる色の数が異なりますので，プログラムを移植するときに注意が要ります．

N-BASIC では整数，単精度，倍精度の3種類あり，変数名を型指定できましたが，N_{60}-BASIC では数値型が1種類しかないために，数値型宣言の命令である DEFSNG，DEFINT 等の命令や数値型の相互変換の命令である CINT 等の命令もありません．

エラー処理のための命令もすべてカットされています．この部分は，プログラムでエラーさえ出さなければ必要ありませんので，別になくても困りません．

TRON，RENUM，AUTO 等のコマンドもなくなっていますが，これらはプログラムのエディット時およびデバッグ中にしか使いませんので，特になくても困りません．

けずられた命令で1番困るのは，ゲーム等で多用する GET@，PUT@で，これらがないために，絵を高速で動かす場合はどうしても機械語を使用することになります．

最後に，N-BASIC のプログラムを N_{60}-BASIC に変換するプログラムについて考えてみましょう．

PC-8001 の上位コンパチブルに設計してあるPC-8801 では，PC-8001 から PC-8801 へのテキストコンバータ（プログラムの自動変換プログラム）は簡単につくることはできますが，

ハード的にまったく異なる PC-6001 と PC-8001 では，実用になるテキストコンバータをつくることは不可能に近いものになります．たとえば BASIC の基本命令だけで書かれたプログラム(ホームコンピュータプログラム集・システムソフト，BASIC COMPUTER GAMES・アスキー等)ならばコンバート可能です．しかし，もしもとになるプログラムが80文字×25行に合わせて画面表示するようにしてあるとすれば，32文字×16行しか表示できない PC-6001 では画面の表示がおかしくなります．結局このようなプログラムは，人手による手直しが必要です．

8-2 いろいろな命令

基本的な命令の説明は入門書等に譲るとして，ここでは N-BASIC ととくに使い方が異なる命令について説明します．

N-BASIC と比較して大幅に異なるのが，画面まわりの命令で，画面出力の命令としては，文字出力関係のPRINT，LOCATE，TAB，SPC等があり，また，グラフィック関係としては，LINE，PSET，PRESET，PAINT 等があります．文字出力の命令は32文字×16行に変更になった以外は，変わる所がありませんのでとくに説明はしません．

ここでは各章で説明をしなかった命令について説明します．

8-2-1 LINE

N-BASIC では画面が40文字と80文字とではX軸の座標のパラメータは 0 ～79から 0 ～159へ変化しましたが，N_{60}-BASIC においては，グラフィックのドットに関係なく 0 ～255(X軸)，0 ～191(Y軸)の固定となっています．そのために，モード 1，2，3 で使用する場合は補正する必要があります．たとえばセミグラフィックモードでは64×48ドットですから，もし，座標(20，10)から(40，20)に直線を書かせる場合は，

```
LINE(20*256/64, 10*192/48)-(40*256/64, 20*192/48)
         X軸の補正     Y軸の補正
```

になります．また，N-BASIC においては PSET，PRESET のどちらかを指定できましたが，N_{60}-BASIC では指定できませんので，点を消したい場合はバックと同じ色を指定することによって行ないます．

またN-BASIC ではキャラクタを使って線をかいたり，箱をかくことが可能でしたが N_{60}-BASIC では，その機能はオミットされています．

```
LINE(0, 0)-(10, 0), "♥"                                N-BASIC の場合
FOR I=0 TO 10 : LOCATE I, 0 : PRINT "♥" ; : NEXT I   N60-BASIC の場合
```

上記のように FOR～NEXT で代用しなければなりません．また，LINE を使用してのブリンク機能もなくなっています．

8-2-2 PSET, PRESET

座標が0～255，0～191で扱われますので，LINE 命令と同じように補正しなければいけません．たとえば，モード2において PSET(0，0)～PSET(3，3)までは同じ位置に点を表示します．なお，PSET のカラーの指定が N-BASIC と異なっています．

PSET(X，Y，C)……N-BASIC

PSET(X，Y)，C……N_{60}-BASIC

座標の補正値を下に示しておきます．

モード	X	Y
1	8	12
2	4	4
3	2	1
4	1	1

図8-1 モードと座標の補正値

使い方
モード2(セミグラフィック)において(20，10)の位置に点を表示するとき

PSET(20*4，10*4)

8-2-3 COLOR

N-BASIC では0～7までの8色を指定できますが，N_{60}-BASIC においてはモードによって指定できる色の種類が異なります．

また，COLOR の第2パラメータと第3パラメータの意味が変わっています．

N-BASIC の第2，第3パラメータに当たる部分が，N_{60}-BASIC にはありません．COLOR に関しては CRTC(画面コントローラの LSI)の違いにより，N-BASIC と N_{60}-BASIC では完全に対応することができなくなっていますので，プログラムの移植には，最も注意を必要とします．

N_{60}-BASICではモードによって色の指定が変わります．

COLOR F，B，C

F=FORE GROUND COLOR	(文字，点の色)
B=BACK GROUND COLOR	(背景の色)
C=COLOR SET	(色の組み合せ)

モード１（テキストモード）

第1パラメータ	第2パラメータ	第3パラメータ	色
1	×	1	緑
2	×	1	緑の反転
3	×	1	オレンジ
4	×	1	オレンジの反転
1	×	2	オレンジ
2	×	2	オレンジの反転
3	×	2	緑
4	×	2	緑の反転

※色は緑とオレンジの２色で第３パラメータによって入れ換わるだけです

図8-1-1

モード２（セミグラフィック）

第1パラメータ	第2パラメータ	第3パラメータ	色
0	×	1	黒
1	×	1	緑
2	×	1	黄
3	×	1	青
4	×	1	赤
5	×	1	白
6	×	1	水色
7	×	1	紫
8	×	1	オレンジ
0	×	2	黒
1	×	2	白
2	×	2	水色
3	×	2	紫
4	×	2	オレンジ
5	×	2	緑
6	×	2	黄
7	×	2	青
8	×	2	赤

図8-1-2

モード3（128×192 カラーグラフィック）

第1パラメータ	第2パラメータ	第3パラメータ	色
1	1	1	緑
2	2	1	黄
3	3	1	青
4	4	1	赤
1	1	2	白
2	2	2	水色
3	3	2	紫
4	4	2	オレンジ

COLORで指定の後，最初のCLS命令で画面全体が第２パラメータの色になる

図8-1-3

モード4（256×192グラフィック）

第1パラメータ	第2パラメータ	第3パラメータ	色
1	0	1	黒地に緑のグラフィック
0	1	1	緑地に黒のグラフィック
1	0	2	黒地に白のグラフィック
0	1	2	白地に黒のグラフィック

図8-1-4

これら COLOR の F と B のパラメータは規定の範囲を超えて指定した場合は最大値，または，最小値にセットされます．ところがモード２のとき，この処理にBUGがあり最大値がうまくセットされません．

モード２のBUG

SCREEN 2, 1, 1 : COLOR 9 : CLS

8-2-4　CLS

N_{60}-BASIC の CLS は，N-BASIC の PRINT CHR$(12)と同じ命令($N_{60}$-BASIC でも PRINT CHR$(12)で画面クリア可能)ですが，N_{60}-BASIC では CONSOLE で指定した範囲でしか画面がクリアされません．

```
10 CLS:CONSOLE 5,5,1
20 FOR I=0 TO 14:LOCATE 0,I:PRINT I:NEXT I:CLS
```

```
 0
 1
 2
 3
 4
Ok
```

```
 10
 11
 12
 13
 14
1 COLOR  CLOAD  GOTO   LIST   RUN
```

実行結果で分かるように CONSOLE で指定した範囲しかクリアされていません．この方が便利な場合が多いので別に困りませんが，もし画面全体をクリアしたい場合は一時的に CONSOLE を元に戻すことで可能です．

N_{60}-BASIC で一番困ることは，グラフィックモードのときに画面に OR でドットを書かせますので一度書かれた上に重ね書きされ，画面が見苦しくなります．

```
10 SCREEN 3,2,2:CLS
20 LOCATE 0,8:PRINT"てすと"
30 LOCATE 0,8:PRINT"あいう"
```

これを解決するには，ドットとバックの色を同じにして塗ればよく，これには LINE の BF 命令を使います．

```
10 SCREEN 3,2,2:CLS
20 LOCATE 0,8:PRINT"てすと"
25 COLOR 1,1,1:LINE(0,96)-(50,107),1,BF:COLOR 2
30 LOCATE 0,8:PRINT"あいう"
```

8-3　色のつけ方

画面に色を付けるには V-RAM およびアトリビュートにデータを書くことによって行ないますが，モードによってその方法が異なりますので，その使い方について説明します．

今，E200H 番地の値が 61H とすると，モードによってどのように表示されるでしょうか．

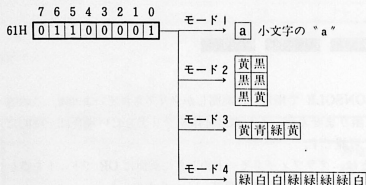



図 8-2　モードと表示の差
※COLORはすべて初期値とする

このように同じデータでもモードによって大きく違うことが分かります．

（1）　モード1

文字とバックの色を変える信号はアトリビュートの bit 1 (CSS) と bit 0 (INV) で行ないます．

CSS 信号が文字の色を，INV がバックの色を決定します．

CSS→0＝緑
→1＝オレンジ
INV→0＝黒
→1＝緑またはオレンジ(CSS できまる)

次のプログラムを実行すれば CSS，INV 信号と色の関係がよく分かるはずです．

```
10 SCREEN 1,2,2:CLS
20 POKE &HE200,&H61:LOCATE 5,8:PRINT "CSS    INV
"
30 FOR I=0 TO 3:A=PEEK(&HE000)AND&HFC:POKE &HE0
   00,A OR I
40 IF I AND 2 THEN LOCATE 6,9:PRINT "1":GOTO 60
50 LOCATE 6,9:PRINT "0"
60 IF I AND 1THEN LOCATE 12,9:PRINT "1":GOTO 80
70 LOCATE 12,9:PRINT"0"
80 FOR J=0 TO 300:NEXT J:NEXT I
90 GOTO 30
```

a

```
CSS    INV
 1      1
```

2 COLOR CLOAD GOTO LIST RUN

（2） モード2

このセミグラフィックは1キャラクタごとにしか色を定義できません．これは PC-8001 の場合とよく似ています．

	D_5	D_6	
D_7 D_6	D_3	D_2	ドットを設定する
色を設定する	D_1	D_0	

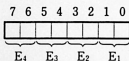
CSS	D_7	D_6	色
0	0	0	緑
	0	1	黄
	1	0	青
	1	1	赤
1	0	0	白
	0	1	水　色
	1	0	紫
	1	1	オレンジ

図8-3　セミグラフィックのカラー設定

```
10 SCREEN 2,2,2:CLS
20 LOCATE 4,8:PRINT"DB 7 6 5 4 3 2 1 0  CSS"
30 FOR CS=0 TO 1:A=PEEK(&HE000)OR CS*2:POKE&HE0
   00,A:LOCATE 24,9:PRINTCS
40 FOR I=0 TO 255
50 C=I:FOR J=7 TO 0 STEP -1
60 IF C>=2^J THEN LOCATE 21-J*2,9:PRINT "1":C=C
   -2^J:GOTO 80
70 LOCATE 21-2*J,9:PRINT"0"
80 NEXT J:POKE &HE200,I:FOR J=0 TO 1:NEXTJ:NEXT
   I:NEXT CS:GOTO 30
```

このプログラムでデータと表示を見ることができます.

(3) モード3

CSSとデータによって色を変えますが，CSSが，1バイトごとでしか定義できませんのでキャラクタ内では4色しか使えません.

この2ドットの組み合わせで色をきめる

CSS	D7, D5 D3, D1	D6, D4 D2, D0	色
0	0	0	緑
	0	1	黄
	1	0	青
	1	1	赤
1	0	0	白
	0	1	水　色
	1	0	紫
	1	1	オレンジ

図8-4　モード3のカラー設定

カラーを調べるプログラム

```
10 SCREEN 3,2,2:CLS:COLOR 2
20 LOCATE 0,8:PRINT"7 6 5 4 3 2 1 0":LOCATE0,10
   :PRINT"CSS"
30 FORCS=0 TO 1:A=PEEK(&HE000) OR CS*2:POKE&HE0
   00,A:COLOR 1,1,1
35 LINE(2,132)-(40,143),1,BF:LOCATE 1,11:COLOR
   2:PRINT CS
40 FOR I=0 TO 255
50 C=I:FOR J=7 TO 0 STEP -1
60 IF C>=2^J THENLOCATE 14-J*2,9:PRINT"1":C=C-2
```

```
    ^J:GOTO 80
 70 LOCATE 14-2*J,9:PRINT"0"
 80 NEXT J:POKE &HE200,I:FOR J=0 TO 100:NEXT
 90 LOCATE 0,9:COLOR1,1,1:LINE(0,108)-(255,119),
    1,BF:COLOR 2
100 NEXT I:NEXT CS:GOTO 30
```

（4） モード4

CSS とデータ・ビットに対応した部分の色が変わります．このモードはプログラム的には一番使いやすいといえます．

CSS	データの各ビット	色
0	0	黒
	1	緑
1	0	黒
	1	白

図8-5　モード4とカラー設定

色を調べるプログラム

```
10 SCREEN 4,2,2:CLS:COLOR 1
20 LOCATE 4,8:PRINT"DB 7 6 5 4 3 2 1 0  CSS"
30 FOR CS=1 TO 2:COLOR,,CS
40 FOR I=0 TO 255
45 COLOR 0,0,CS:LINE(0,108)-(240,119),0,BF:COLO
   R1:LOCATE24,9:PRINTCS-1
50 C=I:FOR J=7 TO 0 STEP -1
60 IF C>=2^J THEN LOCATE 21-J*2,9:PRINT"1":C=C-
   2^J:GOTO 80
70 LOCATE 21-2*J,9:PRINT"0"
80 NEXT J:POKE &HE200,I:FOR J=0 TO 100:NEXT J:N
   EXT I:NEXT CS:GOTO 30
```

このような VRAM の操作は機械語レベルで行なうと，BASIC と比較してきめ細かな操作ができます．

8-4 N_{60}-BASIC にない命令をある命令で代用する

N_{60}-BASIC では削られた命令の内，一部は，他の命令を組み合わせて代用することができます．N-BASIC のプログラムを移植するときの参考にして下さい．

N-BASIC		N_{60}-BASIC
AUTO	→	第 9 章 EXEC と USR 参照
KEY LIST	→	第 9 章 EXEC と USR 参照
BEEP	→	PRINT CHR$(7)または EXEC &H1BCD
BEEP0	→	EXEC &H1B60
BEEP1	→	SOUND 0, 85 : SOUND 1, 0 : SOUND 7, 62 : SOUND 8, 7
MOTOR 0	→	EXEC &H1B49
MOTOR 1	→	EXEC &H1B4B
X MOD Y	→	X－INT(X／Y)＊Y
MON	→	付録参照

第9章　EXECとUSR

第9章　EXEC と USR

9-1　モニタ

PC-6001 にはモニタがありませんので，機械語を使用したいときは PEEK，POKE の命令を用いて，メモリ内容を操作しなくてはなりません．これでは，機械語プログラムをつくるときに非常に不便です．

巻末に，タイニー・モニタプログラムを載せましたが，本格的に機械語を使ったり，BASIC の内部を知りたいときには，やはり，モニタ・ROM が必要でしょう．

暴走するたびに，LOAD していたのではプログラミングどころではありませんし，他の BASIC プログラムを実行することができないからです．

各社から，PC-6001 用のモニタ，アセンブラが発売されていますので，是非，利用されるとよいでしょう．(PAPA-MONITOR・システムソフト，SEAM60-アセンブラ・アスキー　コンシューマー　プログダクツ等)

機械語を打ちこむ簡単なプログラムは下記のとおりです．

```
100 INPUT "ADR";D$
110 D=VAL("&h"+D$):IF D<0 THEN D=65536+D
120 FOR I=D TO 65535:INPUT A$:POKE I,VAL("&h"+A$
    ):NEXT I
```

9-2 EXECとUSR

N_{60}-BASIC においては機械語のプログラムをコールしたい場合は，EXEC または USR を使います．

EXEC と USR の違いは，EXEC が機械語プログラムをコールするだけなのに対して，USR は機械語プログラムにデータを受渡すことができ，また，機械語プログラムから BASIC へデータの受渡しもできます．

さて，EXEC と USR の使い方を実例を交えて説明することにしましょう．

9-2-1 EXEC

EXEC は直接，機械語プログラムに対して直接データの受渡しはできません．

使い方は，EXEC の後に実行する機械語のアドレスを与えます．このアドレスは10進，16進，変数，式のいずれも使用可能です．

機械語のプログラムに RET 命令があると BASIC へ戻ります．このときレジスタの値はどのような値でもかまいません．すなわち，EXEC や USR では BASIC から飛んでくる前にレジスタの退避が済んでいますので，機械語プログラムでレジスタを保存する必要はありません．

```
10 EXEC &H1DFB
```

上のプログラムを RUNして下さい．画面がクリアされたはずです．

これは1DFBH 番地が画面クリア(CLS 命令)の処理ルーチンで，ここをコールすれば当然のことながら画面がクリアされるわけです．

このように EXEC は機械語プログラムをコールするだけで，機械語プログラムとのデータの受渡しはできないようになっていますが，使い方によっては，データを受渡すことが可能です．

それはデータを POKE 文でメモリに書き込んでおき，機械語プログラムに飛ばした後，そのメモリの内容を機械語ルーチンで処理します．その後，結果をメモリにストアし，BASIC に戻り，PEEK 文で取り出すことによって BASIC のデータとして使用する方法です．

例として STICK の命令を EXEC で処理したプログラムを挙げます．なお，ページ指定は32KB，16KB とも 2 を指定しておいて下さい．

```
 10 CLEAR 100,&HDDFF
 20 FOR I=&HDE00 TO &HDE0D
 30 READ DA$:POKE I,VAL("&h"+DA$)
 40 NEXT I
 50 A=0:POKE &HDE10,A
 60 EXEC &HDE00
 70 B=PEEK(&HDE10)
 80 PRINT B
 90 GOTO 50
100 DATA 3a,10,de,cd,39,22,cd,41,07,7b,32,10,de,
    c9
```

50行の変数Aはジョイスティックの番号を表わします.

A＝0：キーボード

A＝1：ジョイスティック1

A＝2：ジョイスティック2

ジョイスティックをお持ちでない方は，必ずA＝0にしておいて下さい.

```
3A10DE   STICK: LD   A,(0DE10H) ジョイスティックのNo.をAccに入れる
CD3922          CALL 2239H      ジョイスティックの方向を調べる
CD4107          CALL 0741H      FAC-1の値を整数に直す
7B              LD   A,E
3210DE          LD   (0DE10H),A Accの値をDE10H番地に格納する
C9              RET
```

PEEK，POKE でのデータ受渡しではデータの数が増えるほど，BASIC での処理に時間がかかり過ぎます．これでは機械語を使う意義がありませんし，テクニック的にも幼稚過ぎます．しかし，工夫しだいで簡単にデータを受渡せることが分かるでしょう．

9-2-2 EXECの応用

N_{60}-BASIC 内部で EXEC がどのように処理されているか説明します．

EXEC 命令があると，まず機械語の飛び先のアドレスを調べて DE レジスタに入れます．次に BASIC テキストのポインタである HL レジスタをスタックに入れます．そして機械語プログラムからの戻り先になる 39B8H 番地をスタックに入れ，DE と HL の値を交換し，HL で指される番地にジャンプします．

例

```
10 EXEC&HDE00:END
```

としたとき，

		10			EXEC	&	H
8400	00	0F	84	0A	00	A5	26	48

	D	E	0	0	:	END		
8408	44	45	30	30	3A	80	00	00

スタック

840CH ②
39B8H ③

① "&HDE00"を整数に直してDEに入れる
② HLの値(840CH番地をさしている)をスタックに入れる
③ スタックに39B8Hを入れる(機械語からの戻り先)
④ DEとHL交換(EX　DE, HL)
⑤ HL番地へジャンプする(JP (HL))

図9-1

となります.

したがって，機械語ルーチン内で，スタック中の HL の値を操作して，EXEC 文で直接パラメータを渡すことができます.

実際のパラメータの与え方としては，EXEC［アドレス］,［パラメータ］の型になります. このような型でEXECを実行すると，スタックに入ったHLの値はカンマのあるアドレスになっていますから，機械語プログラムで，このHLの値を取り出して，後のパラメータを取り込むことによりデータをもらうことができます.

例として行番号を自動的に発生する AUTO のコマンドを作ってみました.

```
 10 CLEAR 300,&HDF3F
 20 FOR I=&HDF40 TO &HDFFF:READ A$:POKEI,VAL("&H
    "+A$):NEXT I
100 DATA E1,21,0A,00,22,F4,DF,22,F6,DF,E1,2B,D7,
    28,26,FE
110 DATA 2C,C2,EA,03,D7,FE,2C,28,09,CD,5F,07,ED,
    53,F4,DF
120 DATA 28,13,CF,2C,28,0F,CD,5F,07,ED,53,F6,DF,
    C2,EA,03
130 DATA 7B,B2,CA,55,07,3E,C3,32,93,FF,32,9C,FF,
    21,C5,DF
140 DATA 22,94,FF,21,AC,DF,22,9D,FF,CD,58,10,CD,
    01,27,CD
150 DATA 2D,27,2A,F4,DF,E5,CD,A1,3A,3E,20,CD,C7,
    26,CD,F9
160 DATA 28,38,2F,D7,CD,1E,05,47,D1,C3,78,04,F1,
    CD,DE,DF
170 DATA ED,5B,F4,DF,2A,F6,DF,19,38,19,11,FA,FF,
    E7,30,13
180 DATA 22,F4,DF,18,CD,F1,3E,0D,CD,C7,26,3E,0A,
    CD,C7,26
190 DATA 18,DE,E1,3E,C9,32,93,FF,32,9C,FF,C3,57,
    04,23,EB
200 DATA 62,6B,7E,23,B6,C8,23,23,23,AF,BE,23,20,
    FC,EB,73
```

```
210 DATA 23,72,18,EC,00,00,00,00,00,00,00,00,00,
      00,00,00
```

使い方

EXEC&HDF40，行番号，増分

行番号に始まり，増分ずつ加算された行番号を発生します．このコマンドはSTOPを押すことにより解除されます．

例　EXEC&HDF40，100，10　　100番から10番おきに行番号を発生．

9-2-3　USR

USRは引数を持って機械語プログラムをコールします．このUSRはN-BASICと比較して2つほど機能が低下しています．

ひとつは，機械語プログラムのアドレスをセットするとき，N-BASICではDEFUSRでセットできたのに対して，N_{60}-BASICではPOKE文を使い，FAEBH番地，FAECH番地にセットしなければならないことです．また，N-BASICでは10種類定義できたのに対して，ひとつしか定義できません．

しかし，このあたりの機能を補なうためにEXECの命令があります．大体において機械語を単にサブルーチンとして使う例が多く，引数を受渡すような使い方は極めて少ないと言えます．それならば定義する必要のないEXECの方がUSRより使いやすいことになります．

さて，もうひとつは受渡す引数の型が数値型しかできないことです．N-BASICでは，数値型(整数，単精度実数，倍精度実数)と文字型が使用可能でした．N_{60}-BASICでは文字型はTM Errorになり，使うことができません．

```
10 CLEAR 300,&HDDFF
20 POKE&HDE00,&HC9
30 POKE&HFAEB,0
40 POKE&HFAEC,&HDE
50 INPUT X
60 A=USR(X)
70 PRINT A:GOTO 50
```

このプログラムは，機械語プログラムではなにもしないでBASICに戻ってきます．これは A＝Xと同じ処理になります．

RUNさせてXにいろいろな値(実数，整数)を入れると，Aの値がXの値と等しいことが分かります．それでは50〜70行を次のように，変更してRUNして下さい．

```
50 INPUT X$
60 A$=USR(X$)
70 PRINT A$:GOTO 50
```

?TM　Error　in　60が表示されてしまいます．N-BASICではこれはエラーになりません．N_{60}-BASICでは機械語から戻ってきたときに数値型であるかチェックしており，文字型はTM　Errorになるように処理されています．それゆえに文字型は使用できないからです．しかし，これもワークエリアを操作すれば，簡単に，文字型も扱えるようになります．

9-3　引数

9-3-1　数値型

引数は，FAC(浮動小数点アキュムレータ)を通して受渡されます．このFACのアドレスはFF65~FF6AHです．

N-BASICにおいては，整数型，単精度型，倍精度型の明確な区別がありましたが，N_{60}-BASICにおいては実数型しかなく，整数型は，実数型に変換されてFACに入ります．そのため，整数型を引数とした場合は実数型から整数型への変換をしなければなりません．その変換ルーチンに当たるのは，N_{60}-BASIC活用表に例としてのっている，0741H番地です．

逆に機械語よりBASICに受渡すときは，整数から実数型へ変換しなければなりません．それが，0D16H番地(A，Bレジスタに値をセットすること)になります．

整数型の例として，エラーコードを引数とし，そのエラーメッセージを出力するプログラムを示します．N_{60}-BASICではエラーコードは0~40までの偶数だけですので奇数は入れないで下さい．

```
 10 CLEAR 300,&HDF00-1
 20 FOR I=&HDF00 TO &HDF1F:READ A$:POKE I,VAL("&
    h"+A$):NEXT I
 30 POKE &HFAEB,0
 40 POKE &HFAEC,&HDF
 50 INPUT X
 60 A=USR(X)
 70 PRINT :GOTO 50
100 DATA cd,41,07,21,9b,03,19,3e,3f,cd,c7,26,7e,
    cd,c7,26
110 DATA 23,7e,cd,c7,26,21,84,03,cd,cf,30,af,32,
    25,ff,c9
```

実数型の引数はそのまま FAC に入るので FAC の内容をそのまま取り出すだけで使えます．2π／Xの計算をさせるプログラムを載せます．

```
 10 CLEAR 300,&HDF00-1
 20 FOR I=&HDF00 TO &HDF0F:READ A$:POKE I,VAL("&
    h"+A$):NEXT I
 30 POKE &HFAEB,0
 40 POKE &HFAEC,&HDF
 50 INPUT X
 60 A=USR(X)
 70 PRINT A:GOTO 50
100 DATA cd,41,39,21,e5,3d,11,66,ff,cd,1a,39,cd,
    06,38,c9
```

9-3-2 文字型

文字型はエラーになることは前に述べました．これは機械語からの戻り先が 0BD8H 番地になっておりここで数値型のチェックをしているからで，この 0BD8H 番地へ，戻るのを機械語の部分で変更すれば文字型を使えるようになります．

```
10 A$="ABCDEFG"
20 C$=USR(A$)
```

上のプログラムを実行すると，FF39H に文字列の長さ，FF3BH，FF3CH に先頭アドレスがセットされます．次は，文字列を引数として渡し，その文字列を逆に CRT に出力するプログラムです．

```
 10 CLEAR 300,&HDF00-1
 20 FOR I=&HDF00 TO &HDF18:READ A$:POKE I,VAL("&
    h"+A$):NEXT I
 30 POKE &HFAEB,0
 40 POKE &HFAEC,&HDF
 50 INPUT X$
 60 A$=USR(X$)
 70 PRINT :GOTO 50
100 DATA 3a,39,ff,2a,3b,ff,06,00,4f,09,2b,47,7e,
    cd,c7,26
110 DATA 2b,10,f9,e1,cd,fa,0a,e1,c9
```

文字型の引数を BASIC に戻す例として，ひらがな→カナ変換のプログラムを挙げます．

```
 10 CLEAR 300,&HDF00-1
 20 FOR I=&HDF00 TO &HDF14:READ A$:POKE I,VAL("&
    h"+A$):NEXT I
 30 POKE &HFAEB,0
 40 POKE &HFAEC,&HDF
 50 INPUT X$
 60 A$=USR(X$)
 70 PRINT A$:GOTO 50
100 DATA 3a,39,ff,2a,3b,ff,47,7e,cd,4f,1a,77,23,
    10,f8,e1
110 DATA cd,fa,0a,e1,c9
```

9-4 BASICを機械語で

9-4-1 ファンクションキー・イニシャライズ

ファンクションキーのイニシャライズ(初期設定)をする場合，N-BASIC では，3A81H 番地より160バイト分ブロック転送するだけの簡単な処理ですみましたが，N_{60}-BASIC ではキー入力の章で説明したようにメモリの節約のために，初期設定の内容が中間言語＋１文字の２バイトになっています．

この２バイトをファンクションキーのバッファに入れるのは，BASIC を使うより機械語の方が簡単にできます．

```
 10 CLEAR 300,&HDF00-1
 20 FOR I=&HDF00 TO &HDF2F:READ A$:POKE I,VAL("&
    H"+A$):NEXT I
100 DATA 06,0a,21,3d,fb,11,67,01,c5,e5,1a,d5,cd,
    65,06,e6
110 DATA 7f,77,1a,13,23,fe,80,38,f6,d1,13,1a,77,
    13,23,af
120 DATA 77,e1,01,08,00,09,c1,10,df,cd,b5,12,c9,
    00,00,00
```

使い方

EXEC & HDF00

9-4-2 KEY LIST

ファンクションキーの内容を見るプログラムは前に BASIC でつくりましたが，それを機械語に置き換えてみます．使い方は KEY LIST とします．

```
key list
F-1 COLOR     F-6 SCREEN
F-2 CLOAD"    F-7 CSAVE"
F-3 GOTO      F-8 PRINT
F-4 LIST      F-9 PLAY
F-5 RUN       F-10 CONT
Ok
```

```
 10 CLEAR 50,&HDF00-1
 20 FOR I=&HDF00 TO &HDF6B
 30 READ A$:POKE I,VAL("&H"+A$)
 40 NEXT
 50 POKE&HFAB1,0:POKE&HFAB2,&HDF
100 DATA FE,97,C2,53,23,D7,E5,11,65,FB,21,3D,FB,
    3E,01,CD
110 DATA 2D,DF,EB,C6,05,CD,2D,DF,F5,3E,0D,CD,C7,
    26,3E,0A
120 DATA CD,C7,26,F1,EB,D6,04,FE,06,20,E4,E1,C9,
    E5,D5,F5
130 DATA 21,69,DF,CD,CF,30,F1,F5,6F,26,00,CD,A1,
    3A,3E,20
140 DATA CD,C7,26,F1,D1,E1,F5,E5,06,08,7E,23,B7,
    28,0B,FE
150 DATA 20,30,02,3E,20,CD,C7,26,10,F0,3E,20,04,
    CD,C7,26
160 DATA 10,FB,E1,01,08,00,09,F1,C9,46,2D,00
```

9-4-3 日本語エラーメッセージ

N_{60}-BASIC のエラーメッセージは 2 文字の省略形で表示されるために，入門用としては使いにくくなっています．その内容を理解するまでは，エラーメッセージ一覧表を頻繁に見ることになり，非常に不便です．

そこでエラーメッセージを日本語で表示させてみましょう．

エラーが発生すると E レジスタにエラーコードをセットして 0401H 番地に飛んできます．そしてこの番地以降のプログラムでエラーを表示しています．エラー処理ルーチンはディスク命令等の追加のためにフック番地を RAM に設けています．それでこのフック番地を利用して，エラーメッセージを変更することができます．

```
 10 CLEAR 300,&HD9FF
 20 FOR I=&HDA00 TO &HDA3F:READ A$:POKE I,VAL("&
    h"+A$):NEXT I
 30 AD=&HDA3E:FOR I=0 TO 20:READ A$,A:C=LEN(A$):
    FOR J=1 TO C
 40 POKE AD+J-1,ASC(MID$(A$,J,1)):NEXT J:POKE AD
    +J-1,A
 50 AD=AD+J:NEXT I
 60 POKE&HFF8D,&HC3:POKE&HFF8E,0:POKE&HFF8F,&HDA
100 DATA cd,f9,34,d5,cd,7a,13,cd,cd,1b,d1,cd,2d,
    27,cb,0b
110 DATA 1c,43,21,3d,da,af,be,23,20,fc,10,fa,e5,
    2a,5d,fa
120 DATA 7c,a5,3c,28,09,cd,a1,3a,21,35,da,cd,cf,
    30,e1,cd
130 DATA cf,30,c3,41,04,ca,de,dd,c6,b5,b2,c3,20,
    00,00,00
200 REM J-ERROR MSG DATA
210 DATA "FORｶﾞ ﾅｲﾉﾆ NEXTｶﾞ ｱﾙ｡",0
220 DATA "ﾌﾞﾝﾎﾟｳ ｱﾔﾏﾘ｡",0
230 DATA "GOSUBﾃﾞ ﾅｲﾉﾆ RETURNｶﾞ ｱﾙ｡",0
240 DATA "DATAｶﾞ ﾀﾘﾏｾﾝ｡",0
250 DATA "ｶﾝｽｳ･ｽﾃｰﾄﾒﾝﾄﾉ ｱﾀｲｶﾞｵｶｼｲ｡",0
260 DATA "ｹｲｻﾝ ｹｯｶｶﾞ ｵｶｼｲ｡",0
270 DATA "ﾒﾓﾘｰ ｶﾞ ﾀﾘﾏｾﾝ｡",0
280 DATA "ｼﾃｲ ｼﾀ ｷﾞｮｳﾊﾞﾝｺﾞｳｶﾞ ｱﾘﾏｾﾝ｡",0
290 DATA "ﾊｲﾚﾂ ｿｴｼﾞ ﾉ ｱﾀｲｶﾞ ｵｶｼｲ｡",0
300 DATA "ﾊｲﾚﾂｦ 2ｶｲ ﾃｲｷﾞｼﾀ｡",0
310 DATA "0ﾃﾞ ﾜﾘｻﾞﾝｼﾀ｡",0
320 DATA "INPUT,DEFFN ﾊ DIRECTﾃﾞﾊ ﾂｶｴﾏｾﾝ｡",0
330 DATA "ﾍﾝｽｳ ﾉ ｶﾀｶﾞ ﾁｶﾞｲﾏｽ｡",0
340 DATA "ﾓｼﾞﾘｮｳｲｷ ｶﾞ ﾀﾘﾏｾﾝ｡",0
350 DATA "ﾓｼﾞ ﾉ ﾅｶﾞｻ ｶﾞ 255ｦ ｺｴﾀ｡",0
360 DATA "ﾓｼﾞ ﾉ ｼｷｶﾞ ﾌｸｻﾞﾂ ｽｷﾞﾏｽ｡",0
370 DATA "CONTﾌﾉｳ｡",0
380 DATA "FNﾊ ﾃｲｷﾞ ｻﾚﾃｲﾏｾﾝ｡",0
390 DATA "ﾃｰﾌﾟ ﾉ ﾖﾐﾏﾁｶﾞｲﾃﾞｽ｡",0
400 DATA "OPERANDｶﾞ ﾀﾘﾏｾﾝ｡",0
410 DATA "ﾌｧｲﾙ ﾉ DATA ﾉ ｶﾀｶﾞ ｵｶｼｲ｡",0
```

第10章　ランダムテクニック

第10章　ランダムテクニック

10-1　TIME

10-1-1　タイム機能

PC-6001は時計用の専用のICを持たないため，1/512secのタイマ割り込みをカウントアップしてそれをタイマがわりに使用しています．

```
10 CLS
20 LOCATE 0,0
30 PRINT TIME
40 GOTO 20
```

上記のプログラムを実行させると下位の桁がめまぐるしく変化しているのが分かります．これはタイマが1/512sec(1.953ms)ごとに2つずつカウントアップしているためです．

PC-6001の内部ではカウンタ用のバッファが4バイト設定されています．TIME関数は，このバッファの内容を10進数に直します．10進では4桁目がほぼ秒の単位になりますが，この機能はあくまでカウンタであって時計の機能ではありません．仮に61秒になっても表示は，101XXXにならずに61XXXとなります．このように時，分，秒を自動的に処理してくれませんので，時計として使用する場合はプログラムで時，分，秒の処理を行なわなければなりません．

またこのカウンタ機能では1000カウントが1000/1024秒(0.9765625秒)になりますので，これを1秒とすると，ゲーム等なら問題ないでしょうが，実用とするなら以下のように補正が必要でしょう．

SECOND＝INT(TIME/1024)

10-1-2　タイマのセット

PC-8001のときはTIME$＝"00：00：00"でタイマを0にセットできましたがPC-6001ではそれができません．ではどうすればカウンタを0にセットできるのでしょうか？

方法1．リセットまたは電源をOFFにして再びONにする．

方法2．カウンタのバッファを0に書き換える．

上記の方法が考えられます．

方法1は非現実的なので方法2について説明します．

カウンタのバッファはFA28～FA2BH番地を使用していますのでここを0に書き換えます．

```
10 PRINT "START";TIME
20 FOR I=&HFA28 TO &HFA2B:POKE I
,0:NEXT I
30 PRINT "SET   ";TIME
Ok
RUN
START 8476
SET    38
Ok
```

20行を実行した後30行で表示されるまで時間がかかるので表示された時間は0になっていません．

機械語で使用するとき，

```
0000 210000   SETIME:LD   HL,0000H
0003 2228FA          LD   (0FA28H),HL
0006 222AFA          LD   (0FA2AH),HL
0009 C9              RET
```

このタイマ機能には，PC-8001より優れている部分があります．それはタイマ割り込みを止めることによってカウントアップを一時停止できるのです．これをうまく使えば累計タイムを測ることが簡単にできます．

```
10 PRINT "START",TIME
20 P=PEEK(&HFA27):P=P OR 1:POKE &HFA27,P
30 OUT &HB0,P
40 PRINT "TIMESTOP",TIME
50 FOR I=1 TO 300:NEXT
60 PRINT "COUNT UP",TIME
70 P=PEEK(&HFA27):P=P AND &HFE:POKE &HFA27,P
80 OUT &HB0,P
```

```
RUN
START               44222
TIMESTOP            44272
COUNT UP            44272
Ok
```

50行はただの時間かせぎです．50行で時間を浪費したのにもかかわらず時間は増えていません．

ポートB0Hがタイマ割り込みのON OFFに使っているので，上記のプログラムでは20行でまずポートB0Hの状態(モータのリレー状態，表示しているページ等)を調べてタイマをOFFにして，ポートB0Hに出力します．そして70行でタイマをONにしてB0Hに出力します．

```
0000 3A27FA    STOPTM:LD    A,(0FA27H)
0003 F601             OR    1               : BIT 0 を 1 にする(タイマー OFF)
0005 D3B0             OUT   (0B0H),A        : ポート出力
0007 3227FA           LD    (0FA27H),A
000A C9               RET

0000 3A27FA    ENTTM: LD    A,(0FA27H)
0003 E6FE             AND   0FEH            : BIT 0 を 0 にする(タイマー ON)
0005 D3B0             OUT   (0B0H),A
0007 3227FA           LD    (0FA27H),A
000A C9               RET
```

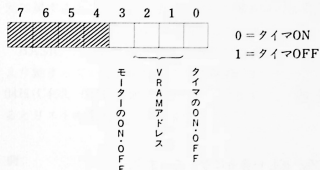
注 ポートB0H

FA27H番地・ポートB0Hのステータスエリア
※DI命令のために正確なカウンタとしては使えない.
※タイマOFFにしたままでいると入力時にカーソルが
表示されず，困ることがある.

図10-1

10-2　知っていればおもしろいランダムテクニック

10-2-1　CLOAD PRINT

セーブしたプログラムのベリファイを行なうには CLOAD？命令を使いますが，CLOAD　PRINT でも，ベリファイを行なうことができます．これは，PRINT の省略形である〝？〟が，中間コード変換において本来の〝？〟と同じものであるとみなされるために起こります．

10-2-2　GOTO

GOTO のつづりは文字と文字との間がいくら空いていても GOTO 命令と判断します．

10　G␣␣O␣␣␣T␣␣␣␣O　100

上のプログラムの LIST をとると

10　GOTO　100

文字と文字とのスペースがつめられて，正しい命令になっています．

GOTO の書き方には一般的には GOTO と GO␣TO があります．N-BASIC では GOTO と GO␣TO の2つ中間言語を持っていて処理をしていましたが N_{60}-BASIC では GOTO だけしかなく，そのかわりに，無条件に文字間のスペースを読みとばすようにして GO␣TO を処理しているために起こる現象です．

10-3 アンリストの方法

LIST を実行してもプログラムリストをとれなくする方法です．

LIST を見せないようにする理由として第一に考えられるのは，プログラムの保護でしょう．

営業用のデモプログラム等の盗用が問題なのはもちろんのこと，個人間でも，自作のプログラムを許可なくコピーされるのは，腹立たしいものです．

アンリストする方法は2つあります．

1.LIST，LLIST のジャンプテーブルを書き換えて，LIST の処理をさせないようにする．

2.最初の行番号を 65535(FFFFH)に書き換える．

1の方法では，EXEC で LIST の処理ルーチンに飛ばすか，LIST のジャンプテーブルを元に戻すことによって，表示される欠点があります．

そこで2の方法について説明しましょう．

BASIC プログラムの先頭行(例のプログラムでは10行)は 8401H 番地(16K 時は C401H)より始まっていることは前の章で説明しました．このときプログラムの先頭の行番号は 8403, 04H(C403, 04H)番地に格納されています．この部分を POKE を使って FFH に書き換えるとリストが取れなくなります．

例

```
LIST

10 REM ｶｹｻﾞﾝ
20 INPUT "A=";A
30 INPUT "B=";B
40 PRINT "A×B=";A*B
50 END
Ok
poke&h8403,&hff:poke&h8404,&hff
Ok
LIST

Ok
```

このアンリストにしたプログラムをSAVE しますと，次に LOAD したときに LIST が見えません．

それを調べてみましょう．

```
csave"カケザ゛ン"
Ok
new
Ok
cload"カケザ゛ン"
Found:カケザ゛ン
Ok
list

Ok
RUN
A=? 256
B=? 16
AxB= 4096
Ok
```

ただしこの方法では問題になることがあります．N_{60}-BASIC では GOTO，GOSUB 命令があると，先頭の行から飛び先のアドレスを探すために，行番号が FFFFH になっていると飛び先がないと判断して，そこで探すのをやめるので，UL Error を出力してしまうのです．

```
10 REM カケザ゛ン
20 INPUT "A=";A
30 INPUT "B=";B
40 PRINT "AxB=";A*B
50 GOTO
Ok
poke&h8403,&hff:poke&h8404,&hff
Ok
RUN
A=? 256
B=? 16
AxB= 4096
?UL Error in 50
Ok
```

これでは困りますのでプログラム中で行番号をもとに戻さなければなりません．ただ単に戻しただけでは STOP キーを押して LIST を見ることができますので，LIST と LLIST の飛び先も変更します．

```
10 REM カケザ゛ン
12 POKE&H8403,10:POKE&H8404,0
14 POKE&HFA8F,&HE0:POKE&HFA90,7
16 POKE&HFA91,&HE0:POKE&HFA92,7
20 INPUT "A=";A
30 INPUT "B=";B
40 PRINT "AxB=";A*B
50 GOTO 10
```

```
poke&h8403,&hff:poke&h8404,&hff
Ok
LIST

Ok
RUN
A=? 256
B=? 16
AxB= 4096
A=?
Break in 20
Ok
LIST
Ok
```

市販のゲームの一部には，もう少し高度なことを行なっているものがあります．

たとえば，プログラムの途中(REM 文の後など)の行番号を FFFFH にし，そのすぐ後に STOP キーのキャンセルを行ない，続いて LIST，LLIST の飛び先を変更する，などというものです．次のプログラムを入力し，POKE &H851A, &HFF：POKE &H851B &HFFを実行して下さい(SAVE するのを忘れずに)．

```
 10 REM *************************
 20 REM *    SAMPLE PROGRAM     *
 30 REM *          of           *
 40 REM *        UNLIST         *
 50 REM *       (C) 1982        *
 60 REM *          by           *
 70 REM *   ASCII SYSTEMSOFT    *
 80 REM *  written by FUJIKEN   *
 90 REM *************************
100 POKE&H851A,&H64:POKE&H851B,0
110 CLEAR 50,&HDF00-1:GOSUB 300
120 POKE&HFA8F,&HE2:POKE&HFA90,7
130 POKE&HFA91,&HE2:POKE&HFA92,7
200 SCREEN 3,2,2:COLOR 2,1,2:CLS
210 RX=INT(RND(1)*5)+1:RY=INT(RND(1)*5)+1
220 PSET(128,186)
230 IF RY>RX THEN A=RY:B=RX:GOTO 250
240 A=RX:B=RY
250 C=INT(A/B):D=A-C*B:IF D<>0 THEN A=B:B=D:GOTO
     250
260 FOR I=0 TO 6.35/B STEP .1/B
270 LINE-(120*SIN(RX*I)+128,90*COS(RY*I)+96)
280 NEXT
285 IF INKEY$="" THEN 285
290 GOTO 200
300 FOR I=&HDF00 TO &HDF26:READ A$:POKE I,VAL("&
    H"+A$):NEXT
310 EXEC&HDF00:RETURN
320 DATA 21,0D,DF,22,02,FA,21,18,DF,22,14,FA,C9,
    C5,06,00
330 DATA F5,CD,78,0E,FE,03,18,09,C5,06,01,F5,CD,
    78,0E,FE
340 DATA FA,CA,2D,0F,C3,BC,0E
```

ところで，エラーのある行の行番号をFFFEHとして実行させせると，どのようなことが起こるでしょうか？　試してみると興味深いでしょう．

```
LIST

10 TECH-KNOW
Ok
poke&H8403,&Hfe:poke&H8404,&Hff
Ok
LIST

Ok
RUN
```

10-4　SCREENのもう一つの使い方

SCREENはディスプレイモードの設定および，画面の切り換えでしたが，それ以外にSCREEN関数と言われる使い方があります．

N_{60}-BASIC活用表にあるように，SCREEN関数はSCREEN(X, Y)の形で使われ，画面上の座標(X, Y)の位置にあるキャラクタを調べます．

```
LIST

10 LOCATE 10,10:PRINT CHR$(65)
20 PRINT SCREEN (10,10)
Ok
RUN




                A
 65
Ok
```

プログラム例を見れば分かるように"A"の文字を画面に書いた後SCREEN関数で呼びだすと，"A"のキャラクタ・コードである65(41H)が表示されます．このSCREEN関数で求められる値は，キャラクタ・コード表と同じ値になります．たとえば"月"であったとすると値は1になります．

```
10 CLS:AD=&H8200:FOR I=0 TO 255
20 POKE AD+I,I:NEXT
30 FOR Y=0 TO 7:FOR X=0 TO 31
40 LOCATE 0,10:PRINT "X=";X;"   Y=";Y;"   ";SCR
   EEN (X,Y);"   "
50 NEXT X,Y
```

このプログラムを実行すると全キャラクタを表示し，SCREEN関数でその値を表示します．

```
月火水木金土日年円時分秒百千万π┴┬┤├┼│─┌┐└┘×大中小
 !"#$%&'()*+,-./0123456789:;<=>?
@ABCDEFGHIJKLMNOPQRSTUVWXYZ[¥]^_
 abcdefghijklmnopqrstuvwxyz{¦}~
♠♥♣♦○●をぁぃぅぇぉゃゅょっ あいうえおかきくけこさしすせそ
 。「」、・ヲァィゥェォャュョッーアイウエオカキクケコサシスセソ
タチツテトナニヌネノハヒフヘホマミムメモヤユヨラリルレロワン゛゜
たちつてとなにぬねのはひふへほまみむめもやゆよらりるれろわん

X= 31    Y= 7     255
Ok
```

10-5 画面を消して実行速度アップ

画面表示のために実行時の約½は, CPU がストップしていることは第3章で説明しました. 計算をさせているときやソート中のときなど, 画面を表示する必要のないときは, 画面の表示を止めると実行速度がアップします.

止めるときには OUT &H93,2を実行し, 戻すときには OUT &H93,3 を実行します.

画面を表示したとき

```
LIST

20 T1=TIME
30 FOR I=0 TO 5000
40 S=S+1
50 NEXT
60 PRINT TIME-T1
Ok
RUN
 44226
Ok
```

画面を消したとき

```
LIST

10 OUT&H93,2
20 T1=TIME
30 FOR I=0 TO 5000
40 S=S+1
50 NEXT
60 PRINT TIME-T1
70 OUT&H93,3
Ok
RUN
 22248
Ok
```

上のプログラムによると, 実行速度が約2倍になっていることが分かります.

10-6　1行は71文字以上可能か？

1行は71文字までですが，これはあくまでキーバッファが71文字分しか確保されないためで，そのため，普通はLISTをとると1行が71文字以内になってしまいます．ところが，ある特定の場合のみLIST時で71文字以上出力されることがあります．

```
10 ?:?:?:?:?:?:?:?:?:?:?:?:?:?:?
:?:?:?:?:?:?:?:?:?:?:?:?:?:?:?:?
:?:?"1"
LIST

10 PRINT:PRINT:PRINT:PRINT:PRINT
:PRINT:PRINT:PRINT:PRINT:PRINT:P
RINT:PRINT:PRINT:PRINT:PRINT:PRI
NT:PRINT:PRINT:PRINT:PRINT:PRINT
:PRINT:PRINT:PRINT:PRINT:PRINT:P
RINT:PRINT:PRINT:PRINT:PRINT:PRI
NT:PRINT"1"
Ok
```

上のプログラムで分かるようにPRINTの省略形である〝?〟を使ったときに71文字以上出力されます．

キー入力されたプログラムは中間言語に変換されます．このときにキー入力されたコマンドは1番短かいものでも2文字(IF, TO等)です．これを1バイトの中間言語に置き換えますから，中間言語に変換されたあとは，キー入力された文字数よりも短くなっています．LISTするときに中間言語をコマンドに変換しますので，キー入力された長さと同じになります．

ところが〝?〟を使ったときは，〝?〟とPRINTの省略形の中間言語は同じ1バイトですので，中間言語に変換されても長さは変わりません．そしてLIST時に〝PRINT〟に直すために1行が71文字以上出力されるのです．

反対に，キー入力は71文字以内であるのに，LISTをとるとその途中までしか受け付けられていない場合もあります．

```
10 PRINT "日月火水木金土日月火水木金土日月火水木金土日
月火水木金土日月火水木金土日月火水木金土日月火水木金土日月火水木
金土日月火水"

LIST

10 PRINT "日月火水木金土日月火水木金土日月火水木金土日
月火水木金土日月
Ok
```

71文字キー入力したのに，LISTをとると40文字しか表示されておらず，後ろがすべてカットされています．これはグラフィックキャラクタが14H＋キャラクタコードの2バイトで表わすために，キーバッファにはグラフィックキャラクタを1文字入力するごとに2バイトず

つ使用して起こる現象です．グラフィックキャラクタを1行に多用するときは，文字の欠落に注意する必要があるでしょう．

10-7　拡張ROMエリアの使い方

4000～7FFFH番地は拡張ROMエリアになっており，ここにユーザーがつくったプログラムをROMに入れて使用することが可能です．また，電源ONとともに拡張ROMのプログラムを走らせることもできます．

4000H番地に41H(″A″)，4001H番地に42H(″B″)が書き込んであると4002, 03H番地に書かれているアドレスをコールします．

例

4000H	41H	5000番地を **CALL** する
1	42H	
2	00H	
3	50H	

N_{60}-BASICでは初期設定，画面クリア，ファンクションキーの表示を終了したときに，この4000, 01H番地を調べます．そして″A″，″B″が書いてあれば次の処理先である0084H(″How Many Pages?″を出力する部分)をスタックに入れます．

機械語だけのプログラムならば，問題はありませんが，BASIC＋機械語，あるいは内部ルーチンを多用しているプログラムではページ数の設定を行なう必要がある場合もあります．

この部分はN_{60}-BASICの内部を理解していませんと使いこなすことができません．

10-8 PRINTとLPRINTの切り換え

画面出力のプログラムをそのまま使ってプリンタに出力したいとき，あるいはPRINTをLPRINTに書き換えたいときがあります．LISTを表示してPRINTの前に〝L〞を1つずつ付け加えるのも1つの方法ですが，これではあまりにも幼稚過ぎます．これ以外の方法をいくつか挙げてみましょう．

1.プログラムをつくる時点であらかじめPRINTを使わずにPRINT#-Xにしておいて変数Xの値を0または3にしてCRTとプリンタを切り換える．

2.CRTとプリンタの切り換えのフラグをあらかじめセットしておいて，インタプリタ内の一文字出力処理で振り分ける(PC-8001のPRINT，LPRINTの切り換えはこの方法を用いているのが多い)．

3.RAM上に命令のジャンプテーブルがあるのを利用してPOKE文で直接PRINTとLPRINTの飛び先を入れ替える．

このような方法があります．2の方法ですと若干ですが機械語のプログラムが必要です．

ここでは3の方法について説明します．LPRINTは087AH番地，PRINTは087EH番地にその処理プログラムがあります．また，PRINTのジャンプテーブルはFA8B,8CHになっています．この番地には087EH番地がセットされています．これを087AH番地に書き換えますとPRINT命令でありながらLPRINT命令の働きをします．

```
 10 PRINT "1──CRT  2──プリンタ"
 20 A$=INKEY$:IF A$=""THEN 20
 30 IF A$="1" THEN GOSUB 200:GOTO 100
 40 IF A$="2" THEN GOSUB 300:GOTO 100
 50 GOTO 20
100 PRINT "0123456789":GOSUB 200:GOTO 10
200 REM PRINT
210 POKE &HFA8B,&H7E:POKE &HFA8C,8
220 RETURN
300 REM LPRINT
310 POKE &HFA8B,&H7A:POKE &HFA8C,8
320 RETURN
```

GOSUB 200でCRTに，GOSUB 300でプリンタに切り換わります．

10-9 PEEK, POKEを使って省メモリ化

メモリを直接読み書きするPEEK, POKE命令はゲームで使う以外に実用ソフトで，おもしろいことができます．

たとえば成績処理プログラムにおいて生徒の点数を配列に入れるのが普通の使い方ですが，これですと1教科について5バイトずつメモリを使用します．テストの点数は0～100までで表わされます．そうすると点数に関しては5バイトの実数型を使う必要はないわけです．1バイトで0～255まで扱えるPEEK, POKEを使用することによって5倍のデータを扱うことができます．

次に2次元の配列をPEEK, POKEでつくった場合の例を示します．

```
   10 XMAX=100:YMAX=10:AD=&HD000
10000 DA=PEEK(XMAX*Y+X+AD):RETURN
11000 POKE XMAX*Y+X+AD,DA:RETURN
```

XMAX YMAX	添字の最大値
AD	ストアするメモリの先頭アドレス
X,Y	配列の添字
GOSUB10000	メモリよりデータを読み出す。
GOSUB11000	メモリにデータをしまう。

10-10 アペンド

複数のプログラムをアペンド(結合)したい場合があります．方法としては2とおりあります．まず1つめはただ単にプログラムⒶの後ろにプログラムⒷを付け加えるもので，2番目の方法はPC-8001ではMERGEと呼ばれる方法でプログラムⒶとⒷとを結合させ，もし同じ行番号があれば後からLOADしたⒷの内容が残ります．また，ⒶとⒷに同一の行番号がない場合には重ね合わせる形で結合します．

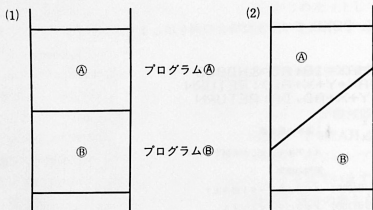



図10-2 アペンド状態

N_{60}-BASICには行番号を変更する命令がありませんので，このような完全なアペンド機能をつくることができません．

それでは1の方法について説明します．

プログラムA

```
10 PRINT "PROGRAM A"
20 FOR I=1 TO 10 :NEXT
```

プログラムB

```
100 PRINT "PROGRAM B"
110 GOTO 10
```

プログラムⒶにⒷを，結合する場合，すでにプログラムⒷはカセットにSAVEされているものとします．

1.まずプログラムⒶを入力(キー入力またはLOAD)します．

```
LIST

10 PRINT "PROGRAM A"
20 FOR I=1 TO 10:NEXT ——————————1
Ok
a=peek(&hff56)+peek(&hff57)*256- ————2
2
Ok
poke&hfa5f,a-int(a/256)*256:poke ————3
&hfa60,int(a/256)
Ok
```

2.FF56,57H番地(変数領域開始番地)の内容を読み出して,プログラムⒶをキー入力した場合は2，カセットからロードした場合には9引く.

3.2で求めた値を FA5F,60H(プログラム開始番地)に下位，上位の順にセットする.

ここで LIST をとると何も表示されないはずです．これはプログラム開始アドレスをプログラムⒶの終了アドレスに変更しているためです.

```
LIST
Ok
cload
Found:B ———————————————————4
Ok
LIST

100 PRINT "PROGRAM B"
110 GOTO  10
Ok
```

4.プログラムⒷを LOAD します．LIST をとりますと，プログラムⒷが表示されます.

```
poke&hfa5f,1:poke&hfa60,&h84 ————————5
Ok
LIST

10 PRINT "PROGRAM A"
20 FOR I=1 TO 10:NEXT
100 PRINT "PROGRAM B"
110 GOTO 10
Ok
```

5.FA5F,60H(プログラム開始アドレス)を元に戻します．LIST をとりますと，ⒶとⒷが結合されています.

注） RAM が 16KB の場合は次のようにして下さい.

```
poke&hfa5f,1:poke&hfa60,&hc4
Ok
```

10-11　行番号を0にする方法

時々，マイコン雑誌に発表されたプログラムや市販されているゲームプログラムで，一部行番号を0にしてあるのを見かけます．

```
0 REM ***************************
0 REM * PC-6001                 *
0 REM *    Graphic Mahjong      *
0 REM *                Ver1.3   *
0 REM * program                 *
0 REM *   by Ray Kazuto         *
0 REM * sound & title design    *
0 REM *    by  Yellow Panther   *
0 REM ***************************
```

このようにする深い意味はないのですが，0 REM(ゼロレムと呼んでいます)にすると，なんとなくプログラムが高級に見えます．

このプログラムは BASIC で簡単につくることができます．

行番号を0にするプログラム

```
65000 INPUT "No.";NO:CU=PEEK(&HF
A5F)+PEEK(&HFA60)*256
65010 LN=PEEK(CU+2)+PEEK(CU+3)*2
56
65020 IF NO<LN THEN END
65030 POKE CU+2,0:POKE CU+3,0
65040 CU=PEEK(CU)+PEEK(CU+1)*256
:GOTO 65010
```

上記のプログラムを行番号を0にしたいプログラムの後に付け加えます．そして RUN 65000 とすると"No."と表示されますから，変更したい所までの行番号を入れます．しばらくすると"Ok"と表示されますので，LIST をとると行番号が0になっています．

例

```
   10 REM ***************************
   20 REM * PC-6001                 *
   30 REM *    Graphic Mahjong      *
   40 REM *                Ver1.3   *
   50 REM * program                 *
   60 REM *   by Ray Kazuto         *
   70 REM * sound & title design    *
   80 REM *    by  Yellow Panther   *
   90 REM ***************************
65000 INPUT "No.";NO:CU=PEEK(&HFA5F)+PEEK(&HFA60)*
      256
65010 LN=PEEK(CU+2)+PEEK(CU+3)*256
65020 IF NO<LN THEN END
```

```
65030 POKE CU+2,0:POKE CU+3,0
65040 CU=PEEK(CU)+PEEK(CU+1)*256:GOTO 65010
```

100行までを0にする場合

```
run 65000
No.? 100
Ok
LIST

0 REM **************************
0 REM * PC-6001                *
0 REM *    Graphic Mahjong     *
0 REM *               Ver1.3   *
0 REM * program                *
0 REM *    by   Ray Kazuto     *
0 REM * sound & title design   *
0 REM *    by   Yellow Panther *
0 REM **************************
```

10-12 PRESETをPSETとしても使える

PSETではX，Y座標とカラーを指定しますがPRESETでもカラーの指定ができます．

PRESET(10, 10), 2

エラーにならずに点を表示します．これはPSET(10, 10), 2と同じ機能になっています．

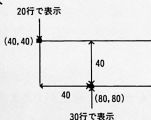
N_{60}-BASICインタプリタでは，PRESETの処理はAccに背景の色をセットし，PSETの処理ルーチンに飛ばしているため，カラー指定ができるわけです．

10-13　グラフィックで相対座標が使える

グラフィック命令では，絶対座標を指定して，点や線を表示しますが，STEP を使うことによって，相対座標で使うこともできます．

この機能は1つ前の座標を原点として，STEP で指定した値を加算した値を次の座標とします．

```
10 SCREEN 2,1,1:CLS
20 PSET(40,40),1
30 PSET STEP(40,40),2
```

このプログラムを実行してみて下さい．点が2個表示されます．

```
1 COLOR CLOAD GOTO LIST RUN
```

20行で表示された点の位置から(40, 40)加算された位置に点が表示されています．この相対座標の指定には当然のことながらマイナスの数も使えます．

```
PSET STEP(-40,-40),4
```

この相対座標の指定は PSET の他，PRESET，LINE，PAINT，POINT 命令でも使うことが可能です．

例　**LINE**

```
10 SCREEN 2,1,1:CLS
20 LINE(80,40)-STEP(40,40),2
```

```
Ok
```

```
1 COLOR CLOAD GOTO  LIST  RUN
```

```
10 SCREEN 3,2,2 :CLS
20 PSET(0,0),2
30 FOR I=1 TO 10
40 LINE STEP(6,3)-STEP(4,2),2
50 NEXT
```

10-14 エラーの音を変えてみよう

サウンド機能のちょっとおもしろい利用法です．

N_{60}-BASIC のエラー処理ルーチンはワークエリア内に 2 ケ所フックを持っています．これを利用して遊んでみましょう．

エラーを出すと〝ピッ〞という音が出ますが，これではあまりに単調なので，エラーを出すとヒューという音がして，音楽まで流れます．(いささかうるさいという気もしますが…)．

次のプログラムを間違いなく入力して RUN します．OK が表示されたら，もうこのプログラムは不要ですから NEW を実行してかまいません．リセットボタンを押すまでは，エラーを出すたびにしつこく音を出します．リセットボタンを押してしまったときは，プログラムの最後の 4 行の POKE 文をダイレクトで実行してください．また復活します．

電源を切ってしまったら仕方がありません．また同じプログラムを入力してください．

また，このプログラムは，機械語を使用していますから，とくにデータの部分は間違いなく入力してください．また危険防止のため入力が終わったら，必ずカセットに SAVE してから RUN させましょう．

```
 10 REM************************
 20 REM*  NEW ERROR SOUND      *
 30 REM*       for PC-6001     *
 40 REM*     by YELLOW PANTHER *
 50 REM************************
 60 CLS:PRINT"Now writing machine language";
 70 FOR ADDRESS=&HDF00TO&HDF7A
 80 READ DA$:POKE ADDRESS,VAL("&h"+DA$):PRINT"."
    ;
 90 NEXT ADDRESS
110 DATAf5,c5,d5,cd,b3,1b,3e,08
120 DATA5f,cd,c5,1b,3e,01,1e,00
130 DATAcd,c5,1b,7b,1e,78,cd,c5
140 DATA1b,06,06,c5,06,0a,1d,cd
150 DATAc5,1b,c5,06,ff,10,fe,c1
160 DATA10,f4,06,14,1c,cd,c5,1b
170 DATAc5,06,ff,10,fe,c1,10,f4
180 DATAc1,10,e0,cd,b3,1b,21,4f
190 DATAdf,cd,b3,1e,3a,1b,fd,b7
200 DATA20,fa,d1,c1,f1,c9,00,22
210 DATA54,32,30,30,4f,53,31,33
220 DATA4d,31,30,30,4c,38,44,34
230 DATA44,2e,44,31,36,44,34,46
240 DATA2e,45,31,36,45,2e,44,31
250 DATA36,44,2e,43,2b,31,36,44
260 DATA34,22,00
270 REM HOOK INITIALIZE
290 POKE&HFF8D,&HCD
300 POKE&HFF8E,&H00
310 POKE&HFF8F,&HDF
320 POKE&HFF90,&HC9
330 PRINT"Complete!"
```

10-15 SOUND, PLAY 関係のデフォルト値

PC-6001 の電源投入時の PSG の値は次のようになっています.

レジスタ番号	データ
0	0
1	0
2	0
3	0
4	0
5	0
6	0
7	56
8	0
9	0
10	0
11	0
12	0
13	0

本によってはレジスタ8, 9, 10のデータが違う値になっているものもありますが, これは0が正解のようです.

また PLAY 命令の MML のそれぞれのデフォルト値は次のようになっています.

MML	データ
V	8
L	4
Ō	4
S	1
M	255

参考にしてください.

付録

付-1 I/O ポート一覧表

ポートアドレス 10進	16進	内　容
128	80H	(入出力)μPD8251(USART)データワード } RS-232C用
129	81H	(入出力)μPD8251(USART)コントロールワード } RS-232C用
		(NEC μPD8251データシート参照のこと)
130 ∫ 143	82H ∫ 8FH	80H, 81Hのイメージ
144	90H	(入出力)ポートA　μPD8049とのコミュニケーションに使用 (モード2で使用) } μPD8255
145	91H	(出　力)ポートB　プリンタデータ(モード0で使用) } μPD8255
146	92H	(入出力)ポートC } μPD8255

ビット	7	6	5	4	3	2	1	0
入出力	入力	出力	入力	出力	入力	出力	出力	出力
内　容	8049	8049	8049	8049	8049	CG 切り換え	CRT KILL	プリンタストローブ

ビット7〜3：モード2　ビット2〜0：モード0

0=CGON　0=KILL
1=CGOFF　1=表示

10進	16進	内　容
147	93H	8255 モードセットおよびポートCのビットセット，ビットリセット } μPD8255
148 ∫ 159	94H ∫ 9FH	90H〜93Hのイメージ
160	A0H	PSGレジスタアドレスラッチ } AY-3-8910
161	A1H	PSGライトデータ } AY-3-8910
162	A2H	PSGリードデータ } AY-3-8910
163	A3H	PSGインアクティブ } AY-3-8910
164 ∫ 175	A4H ∫ AFH	A0〜A3Hのイメージ
176	B0H	(出　力)システムラッチポート　ワークエリア(0FA27H)

ビット	7	6	5	4	3	2	1	0
入出力					出力	出力	出力	出力
内　容					リレー	V-RAMアドレス切り換え		タイマ割り込み

0=OFF　00=C000H　0=ON
01=E000H
1=ON　10=8000H　1=OFF
11=A000H

177 ∫ 191	B1H ∫ BFH	B0Hのイメージ
192	C0H	(入　力)プリンタ用入力ポート(bit 1のみ使用) 0＝BUSY 1＝READY
193 ∫ 207	C1H ∫ CFH	C0Hのイメージ
208 ∫ 211	D0H ∫ D3H	フロッピーディスク　D0H(8ビット)……………入力 D1H(8ビット)……………出力 D2H(上位4ビット)………出力 (下位4ビット)………入力　} μPD8255 D3H μPD8255モードセット
	D4H ∫ DFH	D0H~D3Hのイメージ

付-2　N_{60}-BASIC インタプリター覧表

〔用語解説〕

DW　　アセンブラの疑似命令でワード定数を示すもの(DEFW)

FAC　　浮動小数点アキュムレータのこと．FF66～FF6AH までの FAC を FAC①，FF6D～FF71H までの FAC を FAC②と呼ぶことにしました．

〔マーク解説〕

○	サブルーチンマーク	ユーザーが簡単に使えそうなサブルーチンを示す．
□	予約語マーク 1	N_{60}-BASIC の予約語の処理ルーチンを示す． このマークが付いているルーチンは下記のような使い方ができます． PRINT"ABC" 内部 [95H], 22H, 41H, 42H, 43H, 22H, 00H (中間言語形式) ↑ このアドレスを HL に入れて RST 10H CALL 087EH (PRINT の処理アドレス) で，PRINT "ABC" が実行されます．(第2章を参照)
☒	予約語マーク 2	N_{60}-BASIC の予約語のうち，□印で示したような使い方ができないもの，無意味なものにこの☒をつけている．
△	ルーチン・マーク	サブルーチンではないが，ユーザーが使えそうなアドレスを示す．

〔注意〕

● コメント文で示した例(ex)の中には，使用方法を誤まると暴走するものもありますから，コメント文の内容だけで理解できる方のみ使って下さい．

● コメント中の A はアキュムレータ，B, C, D, E, H, L はレジスタ，SP はスタックポインタのことです．

Z＝ゼロフラグ　CY＝キャリーフラグ

マーク	アドレス	内　　容	コ　メ　ン　ト
	0000	コールド スタート	ハード・ソフトのイニシャライズ後0040Hへ行く.
	0003	テーブル (DW 0741H)	FAC①の値を DE に入れるルーチンのアドレスが示されている.
	0005	テーブル (DW 0D16H)	A, Bレジの値をFAC①に入れるルーチンのアドレスが示されている.
○	0008	パラメータ チェック	(HL)とRST 8H命令の次のアドレスのデータとを比較し, 違えばSN Errorとなる. パラメータが合えば一文字解析ルーチンへ行き, 比較データの次のアドレスへ戻る. ex) 09EB : RST 8H 09EC : DB 3BH ; ';'(比較データ) 09ED : PUSH HL (HL)と3BHを比較し, 合えば, 09EDHへ戻る.
○	0010	1文字解析	HLを1つ進めた後(HL)の内容をAに入れる. Aの値が, 3AH(コロン)か00HならばZ=1, 数字を表すASCIIコード(30 ～ 39H)ならばCY=1として戻る。
	0018	RST 18H用フック (JP FFDBH)	RST 18Hを実行するとFFDBH番地へ飛ぶ, 初期設定ではFFDBH番地にはRET命令(C9H)があるので, 何もしないが, ユーザーがこれを変更して使うことができる.
	001B	テーブル (DW FAEBH)	USR関数の処理アドレスが格納されているアドレスが示されている.
○	0020	HLとDEの比較	HL>DEのとき　Z=0, CY=0 HL=DEのとき　Z=1, CY=0 HL<DEのとき　Z=0, CY=1 となる. またHL, DEの内容は変化しないが, Aは変化する.
	0026	テーブル (DW FA1FH)	CMTのボーレートを示すフラグのアドレスを示している. (第4章を参照)
○	0028	符号 チェック	FAC①の数の符号を調べる. 負の時　A=FFH　Z=0 0の時　A=00H　Z=1 正の時　A=01H　Z=0 となる.

アドレス	名称	内容
0030	RST 30H 用フック (JP 4030H)	RST 30Hは拡張ROMへ飛ぶようになっているため，ユーザーがそのまま使うことはできない．
0038	RST 38H 用フック (JP FFE1H)	0018H と同様
0040	イニシャライズ2	N_{60}-BASIC ソフトのイニシャライズ・ルーチンの始まり．
0043	ファンクションキー初期設定	第4章参照
006F	拡張 ROM チェック	第10章参照
0084	ページ数設定	How Many Pages? の入力とページ関係データの設定
00D6	初期メッセージ表示	N_{60}-BASIC 〜 Bytes free　の表示を行なう． 表示終了後　BASIC　テキストのエディト(0442H)へ行く．
00F0	メッセージ データ	Bytes free　CR/LF
00FE	データ	ページ数によるフリーアドレスの上位バイトの値 F9H ：ページ1の時 DFH ：ページ2の時 BFH ：ページ3の時 9FH ：ページ4の時
0102	メッセージ データ	How Many Pages?
0111	メッセージ データ	N_{60}-BASIC…………1981 CR/LF
0134 〜 0194	データ	ワークエリアの初期データ FA00〜FA60H まで．
0195 〜 01EA	ジャンプテーブル (コマンド)	各命令の処理アドレスが示されている．これらのデータは，ワークエリア内に移される．第2章の6項を参照．
01EB 〜 021C	ジャンプテーブル (関数)	

	アドレス	名称	内容
	021D 〜 02D4	中間言語名テーブル (コマンド)	キーワードを ASCII 英大文字で格納してある．ただし，それぞれキーワードの最初の1文字だけマスク(ビット7を1にしている)してある． PC-8001 では中間コードも一緒にテーブル内に持っていたが PC-6001 では，最初から何番目に出てきた中間言語かカウントし，それをもとに計算して中間コードを求めている． ex) C5, 4E, 44 ｝END 80H C6, 4F, 52 ｝FOR 81H
	02D5 〜 036D	中間言語名テーブル (関数)	
	036E	中間言語名テーブルの終了マーク	
	036F 〜 0383	演算用テーブル	+, −, *, ／, ∧, AND, OR の順に，その優先度と処理アドレスが示されている． ex) 036FH : [79,] [7E, 36] +の処理アドレスは 367EH で優先度を示す． データは 79H である．
	0384	メッセージデータ	Error
	038B	テッセージデータ	in
	0390	メッセージデータ	Ok CR/LF
	0395	メッセージデータ	Break
	039B 〜 03C4	エラーメッセージデータ	エラー番号順に2バイトづつデータが並ぶ ex) 039BH : 4E, 46 ｝NF エラー 039DH : 53, 4E ｝SN エラー
	03C5	FOR, NEXT RETURN 用サブルーチン	
	03E4	エラー処理	READ 処理中，データのタイプミスマッチによるエラーの処理ルーチン
	03EA	SN エラー	以下このアドレスへジャンプするとエラーが表示され，エラー処理を行なう． ex) EXEC &H03FF とすれば TMエラーとなる．
	03ED	/0 エラー	
	03F0	NF エラー	
	03F3	DD エラー	
	03F6	UF エラー	
	03F9	OV エラー	
	03FC	MO エラー	
	03FF	TM エラー	
	0401	エラー処理	Eにエラー番号を入れてここへジャンプすると，そのエラーの名前を表示し，エラー処理を行なう．

△	0441	EDIT2	POP BC 後 EDIT へ
△	0442	EDIT (ホットスタート)	BASIC のテキストエディト (コマンド待ち)
	0478	BASIC テキスト操作	行の変更追加，削除を行なう
	048F		行の削除
	04A3		1行分あける
	04AE		1行挿入
	04C4		BASIC テキストのポインタ変更 (第2章参照)
	04E5	LIST sub	LIST のサブルーチン LIST 行番号を求める DE に開始行番号, (SP)に終了行番号を入れ，開始行番号の行サーチを行ない戻る．
○	0501	行サーチ	DE に行番号を入れ，コールすると， i) { Z=0, CY=0 } 指定行はない, BCに次の行 HL に次の次の行のアドレス ii) { Z=1, CY=1 } 指定行があり，BC にその行 HL に次の行のアドレス iii) { Z=1, CY=0 } プログラムのどの行よりも指定行が大きい のいずれかで戻る．
○	051E	中間言語変換ルーチン	FEDAH からのバッファの ASCII コードによる BASIC テキストを中間言語に直して FEDAH からのバッファへ入れる． HL に FEDAH を入れて戻る．
□	05D6	LLIST	I/O 出力先をプリンタにして LIST へ
□	05DB	LIST	
○	0665	中間言語をキーワードになおす	A に中間コードを入れてコールすると，A にキーワードの頭文字，DE にその命令のキーワードテーブルでのアドレスを入れ，戻る．
⊠	067E	FOR	
	06EA	BASIC 実行メインルーチン	マルチステートメントおよび行終了のチェックを行ない，行番号をワークエリアに入れて，各命令の処理アドレスへ飛ぶ．
○	072A	1文字解析2	HL を1つ進めてから，HL の内容が20H(スペース)でなくなるまで HL を進め，1文字解析(0016H)と同様の処理を行なう．

○ ○	0 7 3 9 0 7 3 A	整数入力 〃	HL を 1 つ進めて，そのアドレスの内容から式解析を行ない整数値を DE に取りこむ．エラーチェックもある． 073AHからのルーチンはHLを1つ進めないだけ ex) PRINT&H1000 内部 95, 26, 48, 31, 30, 30, 30 ↑ このアドレスを，HL に入れてコールすると，DE に 1000Hが入る．
	0 7 5 5	FC エラー	ここへジャンプすれば FC エラーになる．
	0 7 5 A ≀ 0 7 5 E	データ	数値データ −32768 を示す．
○	0 7 5 F	行番号入力	HL 以降の10進 ASCII 数字列を入力して DE に入れる． これは，行番号の入力に使われているルーチンのため FFFAH 以上は入力されない．
□	0 7 8 1	RUN	
☒	0 7 8 F	GOSUB	
□	0 7 A 0	GOTO	
☒	0 7 B C	RETURN	
☒	0 7 E 0	DATA	C＝3AH REM. DATA 共有ルーチン参照
☒	0 7 E 2	REM	C＝00H REM. DATA 共有ルーチン参照
○	0 7 E 4	REM. DATA 共有ルーチン	HL の内容が，C または 00Hと一致するまで HL を進める．ただしダブルクォーテーション内でのデータは00H以外無視する． ex) { DATAはコロン(3AH)または00HまでHLを進める REMは00Hまで HLを進める．
□	0 7 F 5	LET	
	0 8 0 E	文字変数代入	
	0 8 3 D	数値変数代入	
☒	0 8 4 4	ON	
☒	0 8 6 1	IF	
□	0 8 7 A	LPRINT	I/O出力先をプリンタにしてPRINT
□	0 8 7 E	PRINT	PRINT#のチェックも行なっている．

	0902	"," 処理	
	0926	SPC, TAB 処理	
	0950	";" 処理	
	0955	PRINT#, INPUT#終了処理	
○	0967	−255～0 数値入力	HLを1つ進めてそのポインタより数値を入力し, A に入れる, 値が自然数ならFCエラーへ行く.
	0978	メッセージデータ	? Redo from start CR/LF
	098B	INPUT, READ共通サブルーチン	主にエラーの判定を行なう.
☒	09AB	INPUT	
	09C4	RS-232C 指定 INPUT	
	09CF	CMT 指定 INPUT	
	09E0	キーボード指定 INPUT	
☒	0A09	READ	
	0A0E	INPUT, READ 共有ルーチン	(FF49H)=0 なら INPUT, (FF49H)≠0 ならREAD として処理
	0AC5	メッセージデータ	? Extra ignored CR/LF
	0AD6	READ 用サブルーチン	
○	0AF6	数式解析	HL からの式の解析を行ない, それが数値型でなければ TM エラーとなる. データは FAC①に入っている.
○	0AF9	数値型チェック	数値型でなければ TM エラー
○	0AFA	文字型チェック	文字型でなければ TM エラー
○	0B04	式解析 2	(HL)が D2H('='の中間コード)であることを確認してから式解析へ
○	0B07	式解析 3	(HL)が28H('('の ASCIIコード)であることを確認してから式解析へ このルーチンは主にカッコ内の式解析に使われる.
○	0B09	式解析	HLからの式の解析を行ない, データをFAC①に格納する.

	0B66	>, =, <処理1	
	0B81	式解析サブルーチン	条件にあった処理アドレスへジャンプさせる.
	0BCC	−符号処理	
	0BDD	変数処理	
○	0BEE	英小文字→英大文字	HL の内容が，英小文字なら英大文字にしてAに入れる.
○	0BF8	&H 16進数入力	& で始まれば，&H 以下 16 進数を入力し，そうでなければ 075FH(行番号入力)へ行く．DE に入力したデータが入る.
	0C3B	関数処理	拡張用フック有り，式解析サブから A＝(中間コード−D4H)でコール
	0C99	OR 処理	
	0C9A	AND 処理	
	0CBF	>, =, <処理2	
	0CD1	テーブル (DW 0CD3H)	>, =, <処理2の続きが，0CD3H であることを示している.
	0CD3	>, =, <処理3	
	0CF9	NOT 処理	
	0D10	FAC①←(HL−DE)	HL−DE を行ない，その結果を FAC①に格納する HL, DE, BC, A 変化
○	0D15	整数(A, C)を FAC①に格納	整数の上位1バイトをAに，下位1バイトをCに入れてここをコールすると FAC①にデータが格納される.(レジスタの値は確保されないので注意)
☒	0D22	LPOS	
☒	0D27	POS	
○	0D2B	Aの値をFAC①に格納(整数 0〜255)	Aにデータを入れてここをコールするとFAC①にデータが格納される.
☒	0D30	CSRLIN	

☒	0D3A	DEF	単純変数領域内に下図の形で FN 関数が定義される. 10 DEF FNLM(X)=X+1 CC 4D 0D 84 58 00 00 1文字目 マスク有 (L) / 2文字目 マスク無 (M) / 10 DEF FNLM(X)=X+1 ↑ プログラム内のここのアドレス / 1文字目 マスク無 (X) / 2文字目 マスク無 00 / 0 0 FN関数名 LM / FN関数ポインタ / 引数名 X / ダミー FN関数は数値変数として扱われるため,最後の1バイトがダミーとしてある.(第2章参照)
☒	0D61	FN	
○	0DAF	ダイレクト処理チェック	ダイレクトモードならば ID エラー. A 以外のレジスタは変化しない.
	0DBD	FN 関数名解析	
☒	0DCC	INP	
□	0DD6	OUT	
○	0DE3	1バイト整数入力	HL を1つ進めて式解析後, 結果をA に入れる. ただしその値が, 0~255までででなければ, FC エラーとなる. Eレジにも A と同じものが入っている. ex) OUT 255, 10 内部　90, 32, 35, 35, 2C, 31, 30, 00 ↑ (*) このアドレスを HL に入れてコールするとAには, FFH(10進数の255)が入って戻る.
○	0DE4	1バイト整数入力2	最初に HL を1つ進めない以外 0DE3H と同じ. 例で言えば(*)のアドレスを HL に入れてコールすれば良い.
☒	0DF3	PEEK	
□	0DFA	POKE	
○	0E06	2バイト整数入力	式解析後 DE に整数値が入る. ex) POKE &HA000, 0 内部　94, 26, 48, 41, 30, 30, 30, 2C, 30, 00 ↑ このアドレスをHLに入れてコールするとDEに A000H が入る.
	0E10	PC-6001のイニシャライズ	ワークエリアの設定, ハードのイニシャライズ後 0040Hへ行く

○	0E78	サブ CPU ハンドシェイク 1文字入力	A に8049からのデータが入る.
○	0E8F	サブ CPU ハンドシェイク 一文字出力	A のデータを8049へ送る.
	0EA8	キー割り込み処理①	CMT使用時にSTOPキーを押すと,このルーチンへくる.データは強制的に03H である.
	0EB0	キー割り込み処理②	グラフィックキー,ファンクションキー,STOPキーを押すと,この処理ルーチンへくる.サブ CPU から受取るデータは,グラフィックキーならその ASCII コード,ファンクションキーなら F0~F9H, STOP キーなら FAH となっている.
	0EB5	キー割り込み処理③	一般のキーを押すとこのルーチンへくる.サブ CPU から受け取るデータは,押したキーの ASCII コードである.
	0EB8	キー関係割り込みの共有ルーチン	クリック音の発生.STOP・ESC フラグの設定.バッファへのデータ出力. グラフィックキー・ファンクション キーに14H のマークをつけるのもこのルーチン
	0F31	STOP キー割り込みsub	STOP キーを押すと(CMT 使用時はダメ)このルーチンへきて,PLAY を停止させる. 最初にフック(CALL FFD8H)があるのでSTOP キーのキャンセルができる. ex) FFD8H : JP STOP STOP : POP AF LD A, 07H (STOPキーでベルがなる) JP 0EF6H
	0F40	ゲーム用キー割り込み {STICK STRIG}	{1061H STICK, STRIG データ入力ルーチンを参照}
	0F4A	RS-232C用割り込み	データをバッファへためる バッファフルなら受信を一時中止する.
	0F74	2 msec タイマ割り込み	TIME のカウント カーソルの点滅 PLAY の発生を行なう.
	0F9F	CMT READ 割り込み	FA1DH にデータをセットしデータがきたことを示すフラグをたてる. (FA19H の bit 1 を 1 にする)
	0FB2	割り込み	8049 からの割り込みではないらしい.(未使用)

	0FB7	CMT エラー割り込み	エラーを示すフラグをたてる. (FA19H の bit 4 を 1 にする)
○	0FBC	1文字入力 sub (キーボードから) グラフィック文字は1文字入力の説明を参照	入力待ちがない. キー入力がなければ Z=1で戻り, キー入力があれば A にデータを入れZ=0で戻る. このルーチンは, リアルタイム的1文字入力に使えるが, キーバッファが働いていることに要注意. また, すべてのモードで使用可能. 一文字入力ルーチンのように入力後 TEXT モードになることはない. (A 以外のレジスタの値は保存される)
○	0FC4	1.文字入力 (キーボードから) グラフィック文字は14Hとそれに続けて ASC II コードに 30H をプラスした値で示されるため2回入力しなければならない. ex)月は 14H, 31H である.	入力があるまで, カーソルを点滅させて待つ. 入力があれば, カーソルの点滅を止め, A にデータを入れて戻る. 一文字入力 sub でも述べたが, 入力があると, TEXT モードになるので, グラフィックモード設定時にこのルーチンを使うときは要注意. ページ切り換えキーの操作, ファンクションキーの表示等もこのルーチン内で操作している.
○	0FFF	1文字入力 sub 2 グラフィック文字は1文字入力の説明を参照	1文字入力 sub と同じだが, レジスタの値が確保されないから, ユーザーは1文字入力 sub の方を使った方が良い.
○	103A	キーバッファより1文字入力	キーバッファより A に1文字取ってくる. たまっているデータがなければ Z=1で戻りデータがあれば Z=0で A にデータを入れ戻る.
○	1041	ファンクションキーデータ読み込みルーチン	ファンクションキーカウンタ(FA32H)が, セットしてある場合ファンクションキーのデータを順に取り出して Z=0, A にデータを入れ戻る. カウンタがセットされていない, もしくはデータが0のとき Z=1で戻る.
○	1058	キーバッファクリア	キーバッファをクリアする. ここをコールするだけでバッファに入っているデータがなくなるのでゲーム等のプログラムで使えば有用

○	1061	STICK・STRIGデータ入力 (キーボード用)	サブCPUにコマンド6を送りSTICK・STRIG関係のキー入力状況を調べさせる. サブCPUは調査が済むと割り込みをかける. メインCPU は割り込み処理ルーチンでデータを受取り(FECAH)にセットする. (FECAH)のデータの内容は下図のとおりで, キーが押されたビットが1になる. bit 7 6 5 4 3 2 1 0 (FFCAH) = | スペース | 常に | ← | → | ↓ | ↑ | ストップ | シフト | このルーチンをコールするとデータを A に入れて戻る.
○	1075	CRT 1文字出力 グラフィック文字は14Hに続けて ASCII コードに30Hをプラスした値を送ると表示される.	Aにデータを入れコールすると, カーソル位置より表示する. 14Hのときはグラフィックフラグをたてるだけで次にこのルーチンをコールしたときグラフィック文字としての補正を行なう. コントロールコードは, 07H(CTRL-G), 0AH(CTRL-J), 0BH(CTRL-K), 0CH(CTRL-L), 0DH(CTRL-M), 1CH(→), 1DH(←), 1EH(↑), 1FH(↓)の処理のみ行なわれる.
	108B	14H(グラフィック文字)チェック	
○	10AA	CRT 表示1	A にASCIIコードを入れコールするとカーソルの位値から表示する.
○	10D9	CTRL-G	BEEP 音を発生する.
	10DE	PC-6001 ソフトの初期設定	拡張RAMのチェック, ワークエリアの初期設定等を行なう.
○	116A	カーソル実アドレスセット	FDA8H, A9H に示された X+1, Y+1 のカーソル位置の, モード1におけるV-RAM上のアドレスを求め, FDAA, ABH にセットする.

○	116D	カーソルロケーション	H：X+1，L：Y+1を入れコールするとその位置にカーソルを移動する．1文字表示などを行なえば，指定位置にカーソルが移動したことがはっきりする． X, Y の値の確認は行なわれないので要注意．
○	1179	カーソル SW ON	カーソル表示のフラグをたてる． 全レジスタの値が確保される．
○	1181	カーソル SW OFF	カーソル表示のフラグをクリアして反転の状態なら元に戻す．
○	1191	カーソル反転	テキスト，またはセミグラフィックの場合のみカーソル反転を行なう．
○	11B8	カーソル左端実アドレス算出	Lに Y+1を入れコールすると，モード1における画面左端の V-RAM のアドレスをDEに入れ戻る．DE 以外のレジスタは不変
○	11CD	カーソル実アドレス算出	HにX+1，LにY+1を入れコールすると，モード1におけるその位置の V-RAM のアドレスを HL に入れ戻る．HL 以外のレジスタは不変
○	11DA	1行消去ルーチン	LにY+1，DE にその左端の TEXT アドレス(CALL 11B8H で求められる.)を入れコールすると SCREEN の第2パラメータに応じたページのY行を消す．グラフィックモードでも可能 ex) SCREEN 1，1，1のとき LD L, 01H CALL 11B8H CALL 11DAH RET を実行させると0行が消去される．
	1217	モード3用1行消去ルーチン	
	121F	モード4用1行消去ルーチン	
○	1260	スクロールアップ	Hにスクロール最上行＋1，Lにスクロール最下行＋1を入れコールすると，画面のスクロールアップが行なわれる．全モードで可能 また．Aレジのみ変化
	1268	スクロールアップ，ダウン共有ルーチン	Z=0のときスクロールアップ処理，Z=1のときスクロールタウン処理が行なわれる．

○	12A9	スクロールダウン H：スクロール最下行+1 L：スクロール最上行+1	使い方は，スクロールアップと同じ， ex) SCREEN 1, 1, 1 にて LD HL, 1001H CALL 12A9H RET を実行すると画面が1段下がる.
○	12B5	ファンクションキー表示	画面下段のページ数とファンクションキーの表示を行なうのがこのルーチン.
○	1368	キャラクタ反転	COLOR の第1パラを操作して，キャラクタを反転させる．次に表示されるものが反転される．もう一度このルーチンをコールすればもとに戻る.
○	137A	SCREENをTEXT モードにする.	グラフィック(モード3,4)のときページ1のTEXT モードに SCREEN を設定し直す.
○	138A	SCREEN モードチェック	モード1の時　Z=0　CY=1 モード2の時　Z=1　CY=1 モード3，4の時　Z=0　CY=0 で戻る.
○	1390	SCREEN 第1パラ設定	Aに(第1パラメータ)－1の値を入れコールするとアクティブページのモードを変更する.
○	13DB	アトリビュートデータ算出	Aに(SCREEN 第1パラ)－1の値を入れコールすると COLOR 第3パラを考慮に入れたアトリビュートのデータをAに入れ戻る.
	13E9 ∫ 13EC	アトリビュート基本データ	順にモード1，2，3，4のアトリビュートの基本データが入っている. ex) 13E9H：20H　モード1のアトリビュートデータ
○	13ED	SCREEN　第3パラ設定	A に(第3パラメータ)－1の値を入れコールすると表示ページが切り換わる.
○	140C	SCREEN　第2パラ設定	A に(第2パラメータ)－1の値を入れコールすると，アクティブページが変更される.
	141F	SCREEN　第2パラ設定 sub	アクティブページデータをそのページのワークに退避して，新しいページのデータをアクティブページデータエリアに入れる.
	142A	アクティブページ・データをページ・データエリアへ移す.	SCREEN 第2パラで指定されたページへ移す.

	1 4 3 3	SCREEN　第1パラ設定 sub	モード変更に伴なう COLOR データの受け渡しを行なう．COLOR データを現在のモードの COLOR データ領域へ移し，指定モードの COLOR データをアクティブ COLOR データ領域へセットする．
	1 4 4 9	現在の COLOR データをモードにあった COLOR データ領域へ移す．	
	1 4 5 2	各モード COLOR と各ページデータ領域のアドレス算出ルーチン	
○	1 4 7 8	カーソル実アドレスをグラフィック実アドレスに変換	HL にカーソル実アドレスを入れコールすると，HL にグラフィックモードでの V-RAM のアドレスが入り戻る． HL 以外のレジスタは不変
○	1 4 A 0	グラフィックモード用キャラジェネアドレス計算	A に ASC II コードを入れコールすると，CGROM でのそのコードのデータの開始アドレスを DE に入れ戻る．DE 以外のレジスタは不変
○	1 4 A F	CRT 表示 2	A に ASCII コード HL にカーソルの実アドレスを入れコールすると，その位置に1文字表示を行なう．カーソルの移動は行なわない．
	1 4 C 9	CRT 表示 sub	
	1 4 D 5	テキスト，セミグラモード処理	
	1 4 E 2	グラフィックモード処理	
	1 4 F F	カラーグラフィック sub	
	1 5 2 E	フルグラフィック sub	
	1 5 4 0	スーパーインポーズ	
	1 5 4 9	スクロールアップダウン sub	

	アドレス	内容	備考
	1578 15BA 15C0 15C8 15D1 15D5 15D9 15DD 15E1 15E5	COLOR コマンド0 COLOR コマンド1 COLOR コマンド9 COLOR コマンド2 COLOR コマンド3 COLOR コマンド4 COLOR コマンド5 COLOR コマンド6 COLOR コマンド7 COLOR コマンド8	COLOR 関係のサブルーチン群 COLORコマンドジャンプテーブル参照 COLOR コマンド0，1については不明．また詳細なことについては，まだ解析されていないコマンドもある．
	15E9 ≀ 196E	各 COLOR コマンドの処理ルーチン	COLOR コマンドジャンプテーブル
	196F ≀ 19BE	COLOR コマンドジャンプテーブル	2バイトずつテキスト用，セミグラフィック用カラーグラフィック用，フルグラフィック用の順で各コマンド処理アドレスが示されている．
	196F ≀ 19BE	COLOR コマンドジャンプテーブル	ex）196F：E9,15 コマンド0 テキスト用 15E9H 1971：FB,15 コマンド0 セミグラ用 15FBH 1973：09,16 コマンド0 カラーグラ用 1609H 1975：0D,16 コマンド0 フルグラ用 160DH 1977：0F,16 コマンド1 テキスト用 160FH ⋮

コード ＼ コマンド	0	1	2	3	4	5	6	7	8	9
テキスト	15E9	160F	168F	16AD	1725	17B3	1800	1869	1889	1668
セミグラフィック	15FB	161F	1695	16D5	1744	17D5	181C	187D	18A5	1678
カラーグラフィック	1609	1647	168F	16AD	172F	17BB	1808	1585	18B0	167B
フルグラフィック	160D	1658	169D	16AD	173B	17CD	1814	1885	18B0	168A

コマンドNo.	内容
0	不明
1	不明
2	COLOR 第1パラメータの範囲を調べ，それを越える数について補正を行なう．
3	各COLOR に対応したアトリビュートの値を求める．
4	現在のアトリビュートの COLOR コードを求める．
5	現在の位置より1つ後のアトリビュートでのアドレスを求める．
6	現在の位置より1つ前のアトリビュートでのアドレスを求める．
7	現在の位置より1つ上のアトリビュートでのアドレスを求める．
8	現在の位置より1つ下のアトリビュートでのアドレスを求める．
9	アトリビュートデータのセット（グラフィックモードのときはその色のビットパターンをセット）

（注）コマンド7，8にはまだ不明なルーチンも含まれているので内容が異なる可能性もある．

○	1 9 B F	RS-232C イニシャライズ	RS-232C 用バッファのクリア，8251のイニシャライズを行なう．
○	1 9 D 1	RS-232C 1文字入力	1文字入力するまで待つ，キーボードの1文字入力と同様に，バッファから1文字読んでくるのであって，リアルタイムなものではない． STOP，ESCキーのチェックを行なっているので無限ループに入ることはないだろう．
○	1 9 E 8	RS-232C 1文字出力	A の内容を1文字出力
○	1 9 F D	8251 イニシャライズ	
○	1 A 0 F	LCOPY sub	サーマルプリンタ用グラフィックデータ1バイト出力ルーチン． C にデータを入れコールする．ただしプリンタの方はあらかじめグラフィック出力にしておかなければならない．B, C, A が変化する．
○	1 A 1 C	プリンタ1文字出力	Aにデータを入れコールする．グラフィック文字は無視され，ひらがなはカタカナに変換されて印字される．
	1 A 2 B	プリンタ1文字出力 sub	Aにデータが入り，Cに1が入っていると，BUSY のとき STOP, ESC キーが，チェックされる．C が1でないときは，STOP, ESC キーは，きかなくなる．
○	1 A 4 F	ひらがな→カタカナ変換	Aにデータを入れコールするとひらがなのASCII コードならカタカナのコードに変換して戻る．
○	1 A 6 1	CMT ロード用イニシャライズ	PC-6001 では，SAVE 時と LOAD 時でのイニシャライズ，ストップが同一ではないので注意して欲しい．このルーチンでは8049にCMT(LOAD用)を OPEN することを宣言しモーターを ON にする．
○	1 A 7 0	CMT 1文字入力	Aにデータが入る．Z=0のときは，テープリードエラーでZ=1の時は LOAD に成功している．
○	1 A A A	CMT LOAD 用 STOP	8049にCMT(LOAD用)を CLOSE することを宣言し，モーターを OFF にする．
○	1 A B 8	CMT SAVE 用イニシャライズ	モーターを ON にして8049に CMT(SAVE用)を OPEN する．
○	1 A C C	CMT 1文字出力	Aにデータを入れコールすると出力される．
	1 A E D	CMT SAVE の READY チェック	ここをコールして，Z =0で戻れば準備 OK.

○	1B06	CMT SAVE用 STOP	8049にCMT(SAVE 用)を CLOSE することを宣言し，モーターを OFF にする．
	1B14	CMT 関係コマンドを8049へ送る．	600ボー SAVE OPEN 3DH, 39H 600ボー LOAD OPEN 1DH, 19H 1200ボー SAVE OPEN 3EH, 39H 1200ボー LOAD OPEN 1EH, 19H の2バイトを8049へ送る．
○	1B2A	＊のブリンク表示	CLOADのとき(＊)をブリンクさせる．1回コールすると，(＊)を表示していれば消し，表示していなければ(＊)を表示する．
	1B40	CMT 割り込み用ワークの関係ビットのクリア	(FA19H)の bit 1 と bit 4 を 0 にする．0F9FH, 0FB7Hを参照
○	1B49	モーター OFF	
○	1B4B	モーター ON	
	1B51	2msecタイマ割り込みON	
○	1B54	システムポート(B0H)出力	Bに変更するビットだけ1にしたものを入れAに新しい出力用データを入れコールする．(FA27H)に出力したデータが格納される．
	1B60	PLAY STOP sub	PLAY用ワークとバッファのクリア，PSGの音を止める．ユーザーはここをコールするよりも 1BB3H をコールした方が良い．
○	1BB3	PLAY STOP	PLAY 発生を止める．
	1BB9 ≀ 1BBD	PLAY 用ワークの初期データ．詳細は不明	
○	1BBE	8910(PSG)コントロール1	A にレジスタ No, E にデータを入れコールすると，Dに変更前の指定レジスタのデータが入り，PSG に Eのデータがセットされる．
○	1BC5	8910(PSG)コントロール2	A にレジスタ No, E にデータを入れコールすると PSG にセットする．
○	1BCD	BEEP 発生	CTRL-G 用のルーチン
	1BEC	PLAY 発生ルーチン	2 msec タイマ割り込みで使用 PLAY で作られたデータを，実際に音にして発生させるルーチン．

○	1CA6	ジョイスティック入力	Aにジョイスティックナンバー(01Hか02H)を入れコールするとジョイスティックの状態をAに入れ戻る．A以外のレジスタは不変 bit 7 6 5 4 3 2 1 0 A = ☒ ☒ ☒ トリガー → ← ↓ ↑ ビットが1のときそのボタンは押されている．
□	1CD2	LOCATE	
□	1CF6	CONSOLE	
○	1D6B	1バイト整数入力3	1バイト整数入力2 (0DE4H)と同じだが D, E, B, C のレジスタの値は変化しない．
	1D73	CONSOLE 命令の実行	H に CONSOLE 開始行+1，L に最終行+1が入ってコールされている．(H≦L)
□	1D9B	COLOR	
	1DBB	COLOR 第3パラメータ実行	Aに1, 2を入れコールすると前回と異なるデータならば，画面全体を反転させる．
□	1DF8	CLS	
○	1DFB	CLS 2	ここをコールすれば CLS が行なわれる． ex) EXEC &H1DFB
□	1E04	SCREEN	
	1E58	ページ数チェック付1バイト整数入力	Aに(データ)−1が入り最大ページ数と比較してそれより大きければ FC エラーとなる．
☒	1E65	SCREEN 関数処理	
☒	1E83	TIME	
□	1E9B	SOUND	
□	1EB3	PLAY	
	200C	テーブル (DW 200EH)	PLAY 関係処理ルーチンのジャンプテーブルの先頭アドレスを示す．
	200E ≀ 203B	PLAY 処理ジャンプテーブル	PLAY の" "内での音階，特別の意味を持つ記号等の飛び先を3バイトで示している．音階は ASCII コード+アドレス，その他のコードは ASCII コードにマスクをかけたもの+アドレスで示される． ex) 200E : 41, 1A, 21　音階"A"の処理は 211AH から 2023 : CD, 7B, 20　"M"の処理は 207BH から

	2 0 3 C ≀ 2 0 4 A	音階用データテーブル	詳細は不明
	2 0 4 B ≀ 2 0 6 2	音階データ	C, C#, D, D#, E, F, F#, G, G#, A, A#, B の順で音階周波数の基本データが入っている．
	2 0 6 3	V(音量)処理	
	2 0 7 B	M(エンベロープ周期)処理	
	2 0 9 B	S(エンベロープ形状)処理	
	2 0 A 5	L(長さ)処理	
	2 0 B F	T(テンポ)処理	
	2 0 C C	O(オクターブ)処理	
	2 0 D 9	R(休符)処理	
	2 0 F E	N(数音階)処理	
	2 1 A A	A~G(音階)処理	
	2 2 3 1 ≀ 2 2 3 5	データ 詳細は不明	
☒	2 2 3 6	STICK	
	2 2 4 C	STICKキーボード処理	
☒	2 2 5 6	STRIG	
	2 2 6 E	STRIGキーボード処理	
	2 2 7 6	STICK, STRIG 機器 No 入力	
	2 2 8 6	STICK 方向データ ジョイスティック用	
	2 2 9 6	STICK 方向データ キーボード用	
□	2 2 A 6	LCOPY	

	アドレス	名称	説明
	2 3 4 A ～ 2 3 4 E	LCOPY データI	サーマルプリンタ用グラフィックモード指定 20H　スペース 20H　スペース 0AH　ラインフィード 1DH　グラフィック指定 } C1H　193 } 256×193のグラフィック指定
	2 3 4 F ～ 2 3 5 2	LCOPY データII	セミグラ用のデータ　下図参照 0 0 → 0 0 H 0 1 → 0 F H 1 0 → F 0 H 1 1 → F F H
□	2 3 5 3	KEY	
	2 3 9 3	PLAY 用各処理ルーチンへのジャンプ処理	
□	2 4 7 E	CSAVE	
□	2 4 9 6	CLOAD	
	2 4 E C	LOAD 失敗	
	2 5 0 B	メッセージデータ	Bad　CR/LF
	2 5 1 1	CLOAD 用 file name の格納ルーチン	
	2 5 1 8	CSAVE 用 file name の格納ルーチン	
	2 5 3 9	CMT より file name を格納するルーチン	
○	2 5 4 E	CMT マークチェック	CMTよりEのデータをD回検出するまでループ
○	2 5 6 5	file name の比較	FECBH から6文字と(HL)から6文字を比較して合えば Z=1，違えば Z=0，例外として(FECBH)=0なら Z=1となる．
	2 5 7 6	メッセージデータ	Found :
	2 5 7 D	メッセージデータ	Skip :
	2 5 8 3	file name 表示	
	2 5 9 A	INPUT#−1用 CMT OPEN	
	2 5 A 8	PRINT#−1用 CMT OPEN	

○	25B7	file name のCMT 出力	
	25D0	SAVE メイン部	HLにBASICの開始アドレスを入れてコールする．
	25E5	TIME ディレイ	
	25F6	LOAD メイン部	A＝FFH のとき LOAD A＝00H のとき VERIFY となる． 00H が 10コ検出されると終了で，Z＝0で失敗，Z＝1で成功である．
○	2613	2バイト整数(アドレス)入力	HL の示すアドレスより式解析を行ない，正整数のチェックをしてから，DE にその値を入れて戻る．
□	261D	EXEC	
○	2627	バッファへ1文字送りこむ．	Aにバッファ No, Eにデータを入れコールすると，指定ナンバーのバッファへそのデータを入れる．その際バッファフルならば Z＝1で戻る．
○	2642	バッファから1文字読みだす．	A にバッファNo を入れコールするとAにデータを入れ戻る．Z＝1のときは，バッファにデータがたまっていない場合．
○	266F	バッファマップクリア	A にバッファNo B にバッファの最大容量 3FH DE にバッファの先頭アドレス を入れコールするとバッファをクリアする．
○	26A2	バッファの残りバイト数を求める．	A にバッファNo を入れコールすると HL に残りバイト数を入れ戻る．A にも同じ値が入っている．
○	26B1	バッファマップよりデータを読み出す．	A にバッファNo を入れコールすると，バッファの CからBまで使用していることを示し，HL が，バッファマップの3バイト目を示す．
○	26BB	バッファマップアドレス算出	A にバッファのナンバーを入れコールするとバッファマップの先頭アドレスを HL に入れ戻る．
○	26C7	各I/O 1文字出力	(FA58H)にI/O機器を下図のように指定し，A にデータを入れコールすると指定I/O機器に1文字出力する． (FA58H)＝ 00H CRT 01H プリンタ 02H RS-232C 80H CMT

	26DA	プリンタ1文字出力	
	2701	プリンタのCLOSE	
	271E	CRT 1文字出力	
	2722	CMT 〃	
	2726	RS-232C 〃	
	272A	1文字入力	
○	272D	各I/OへCR/LF出力	A以外のレジスタは不変
○	274D	STOP・ESCキー入力チェック	A以外不変
⊠	2771	INKEY$	
	278E ~ 279D	CRT 1文字出力ルーチン用CTRLコード処理アドレステーブル	CTRL-J, K, L, M[→], [←], [↑], [↓]の順で2バイトずつアドレスが並ぶ.
	279E	[→] 処理	
	27B0	CTRL-M 処理	
	27B5	CTRL-J 処理	
	27C9	[↓] 処理	
	27D9	[↑] 処理	
	27EF	CTRL-K 処理	
	27FA	[←] 処理	
	2874	CTRL-L 処理	
	28BA	SCREEN EDIT 用行フラグのスクロールアップ	Aにスクロール行数, LにY+1を入れコール.
	28CD	SCREEN EDIT 用行フラグのスクロールダウン	
	28E0	SCREEN EDIT 用行フラグ入力	LにY+1を入れコールするとその行の行フラグをAに入れ戻る.
	28ED	SCREEN EDIT 用行フラグセット	LにY+1, Aに行フラグデータを入れコールすると, そのデータを行フラグとしてセットする.

	アドレス	名称	内容
○	28F9	LINE INPUT	コールすると RET もしくは STOP を押すまでSCREEN EDITを行ない，HLに(入力テキストのバッファアドレス)−1を入れ戻る。 STOPを押した時は，CY=1で戻る．
	2905	INPUT用入力ルーチン	
○	2966	コントロールコードサーチ	HLにテーブルの先頭アドレス，Cにデータ数，Aにデータを入れコールすると，Aのデータが，テーブル内にあればZ=1，なければM=1で戻る．
	296E	コントロールコード処理	
	29C2 ∫ 29CD	コントロールコード・データ1 (INSフラグクリア用)	1バイトずつCTRLコードを示す． ex) 29C2H：0D　CTRL-M(CR)を表す．
	29CE ∫ 29D7	コントロールコード・データ2 (CTRL処理用)	1バイトずつCTRLコードを示し，これらは次のジャンプテーブルと逆の順番で対応している．
	29D8 ∫ 29EB	コントロールコード・データ2用ジャンプテーブル	(下表)

データ	CTRL	アドレス	内容
09	I	2A92	TAB 8カラム
0A	J	29EC	LF
08	H	2B12	DEL
12	R	2AC3	INS
02	B	2BAA	カーソル左
06	F	2B86	カーソル右
05	E	2B6C	カーソルより後を消す
03	C	2A82	STOP
0D	M	2A1C	CR
15	U	2B64	1行抹消

	アドレス	名称	内容
	29EC	CTRL-J	
	2A1C	CTRL-M	
	2A82	CTRL-C	
	2A92	CTRL-I	
	2AC3	CTRL-R	
	2ACC	INS中なら1文字分あけるルーチン	
	2ADB	1文字あけるルーチン	
	2B12	CTRL-H	

	2B64	CTRL-U	
	2B6C	CTRL-E	
	2B86	CTRL-F	
	2BAA	CTRL-B	
○	2CA9	英数カナ文字チェック	Aの内容が，0~9，A~Z，a~z，カナのASCIIコードならCY=0ちがえばCY=1となり戻る．
	2CEE	グラフィック関係命令の座標入力ルーチン	STEP処理有り
□	2D37	PRESET	
□	2D3C	PSET	
	2D3F	PSET. PRESET共通ルーチン	
☒	2D55	POINT	
□	2DC7	LINE	
□	2EDC	PAINT	
☒	305B	STR$	
	306B	文字列と文字列データのセット	
	3091	"(ダブルクォテーション)の処理	
○	30CF	メッセージ出力	HLにメッセージの先頭アドレスを入れコールすると，(HL)に00Hがでてくるまで表示する．
○	30E7	文字列追加	Aの数だけ文字列領域に追加
	3102	ガベージ・コレクション	
○	31F3	(BC) → (DE) L個	BCからLバイトDEからへ移す．
○	31FC	文字数チェック	式解析後にコールすると(HL)が文字数を示す．
☒	3229	LEN	
☒	3238	ASC	
☒	3249	CHR$	

☒	3257	LEFT$	
☒	3286	RIGHT$	
☒	328F	MID$	
☒	32BA	VAL	
☒	32DE	FRE	
	32FD	DIM sub	DIM A(10), B(20), などのように宣言する配列が2つ以上のとき","のチェックを行なって DIM のルーチンへ
□	3302	DIM	
○	3307	変数解析 (DIM と共有ルーチン)	HL の示すポインタより1つの変数の解析を行ない. DE にその変数のポインタを入れ戻る. つまり DE は数値変数なら FAC①の数値データの先頭, 文字変数なら文字変数データの先頭を示す.
	3344	FN チェック	
	3355	変数名サーチ	
	3378	新変数登録	
	33BA	配列サーチ	
	33E6	配列サーチ2	
	3413	新配列登録	
○	34A2	メモリチェックとブロック転送	
○	34B0	メモリチェック1	C×2バイトだけの SP 領域の余裕があるか, なければOM エラーとなる.
○	34B9	メモリチェック2	通常のメモリチェックルーチン
□	34CD	NEW	
□	3519	RESTORE	
□	3533	STOP	
□	3535	END	
□	356B	CONT	
○	35A1	英大文字チェック	CY=0で英大文字

□	35A9	CLEAR	
○	35F0	DE ← HL−DE	HL は変化せず
☒	35F7	NEXT	
	367E	＋ 処理	
	3683	− 処理	
	368C	FAC① ← FAC①+FAC②(＋, −共有ルーチン)	
○	3747	4バイト倍長整数加算	FAC①, FAC②の仮数部4バイトを使って FAC① ← FAC①+FAC②を行なう.
○	3759	4バイト倍長整数減算	加算と同様で FAC① ← FAC①−FAC②を行なう.
○	376B	FAC①の補数	FAC① ← FAC①の補数
	37AA	5バイト 2倍化	HL の示すアドレスから5バイトの内容を2倍にして戻す.
	37B4	＊ 処理	
	37EE ～ 37F2	数値データ 0.1	
	37F3 ～ 37F7	数値データ 10	
○	37F8	1/10化	FAC① ← FAC①×0.1を行なう
	3803	／ 処理	
	388E	RST 28H の続き	
☒	3898	SGN	
☒	38B9	ABS	
○	38C3	FAC①の SP 格納	FAC①の内容を下図のように, スタックに退避する. SP→ (FF65H) } ダミー (FF66H) (FF67H) (FF68H) } FAC① (FF69H) (FF6AH)

○	38D5	FAC① → FAC②してから (SP) → FAC①	FAC①の内容を FAC②へ転送後，前記のように格納された FAC の内容を FAC①へ持ってくる．
○	38EA	(SP) → FAC②	前記のように格納されたFACの内容をFAC②へ持ってくる．
	38FC	整数型データ→ FAC①	B C D E B：指数 C：上位 ～ E：下位（C D E：仮数）
	3907	FAC① →整数型データ	
○	3913	(DE)→(HL)へ4バイト転送	
○	3917	(HL)→ FAC② 5バイト転送	
○	391B	(DE)→(HL) 5バイト転送	
○	3939	FAC② → FAC① 5バイト転送	
○	3941	FAC① → FAC② 5バイト転送	
○	3944	FAC① →(HL) 5バイト転送	
○	3981	FAC①より整数型の正数をHLに入れる．(エラーチェック有り)	
○	399B	HL を FAC①に正整数として格納	
☒	39E7	INT	
	3A16	・の処理	
○	3A99	in 行番号表示	in に続けて HL の値を10進表示
○	3AA1	行番号表示	HL の値を10進変換して表示
	3B70 ～ 3B7E	数値データ1 0.5 10000,0000 1E+09	各データについて5バイト使用

	3B7F ～ 3BA2	数値データ2 1×10^8 1×10^7 1×10^6 1×10^5 1×10^4 1×10^3 1×10^2 1×10 1	各データについて4バイト使用
	3BA3	RND	
	3CAF	FAC① → FAC②	
	3CB7	FAC② → FAC①	
	3D03	FAC② →(SP)	
	3D08	FAC① →(SP)	
	3D18	(SP)→ FAC②	
	3D1D	(SP)→ FAC①	
○	3D2D	FAC①の符号チェック	FAC①=0 Z=1 A=0 FAC①>0 Z=0 A=1 FAC①<0 Z=0 A=FF
	3D4A ～ 3E20	数値データ3	
☒	3E21	EXP	
☒	3EA5	LOG	
	3EFA	∧(べき乗)処理	
☒	3F51	COS	
☒	3F57	SIN	
☒	3F92	SQR	
☒	3FD3	TAN	
	3FE5 ～ 3FFF	不使用	

付-3　ワークエリアー覧表

アドレス	内　容
FA00 FA01	DW 0FB2H　割り込み(未使用)
FA02 FA03	DW 0EB5H　キー割り込み③
FA04 FA05	DW 0F4AH　RS-232C 用割り込み
FA06 FA07	DW 0F74H　2 msec タイマ割り込み
FA08 FA09	DW 0F9FH　CMT READ 割り込み
FA0A FA0B	DW 0FB2H　割り込み(未使用)
FA0C FA0D	DW 0FB2H　割り込み(未使用)
FA0E FA0F	DW 0EA8H　キー割り込み①
FA10 FA11	DW 0EA8H　キー割り込み①
FA12 FA13	DW 0FB7H　CMT エラー割り込み
FA14 FA15	DW 0EB0H　キー割り込み②
FA16 FA17	DW 0F40H　ゲーム用キー割り込み
FA18	STOP. ESC キー フラグ　03H＝STOP　1BH＝ESC　00H＝それ以外
FA19	CMT 割り込みフラグ bit1＝1 ならデータ受けとった．bit4＝1 ならエラーが発生
FA1A	・入力先I/O機器指定 00H：キーボード 02H：RS-232C 80H：CMT ・PLAY においてはチャンネルを示す． 03H：チャンネル A 04H：チャンネル B 05H：チャンネル C
FA1B FA1C	DW FB8FH　ラウンドロビン バッファ制御用MAPの先頭アドレスを示す．
FA1D	CMT READ 割り込み用ワーク入力したデータが入る．

FA1E	CLOAD '*' の点滅用フラグ前の状態が入る．
FA1F	CMT ボーレート指定　　00H＝600 ボー　FFH＝1200 ボー
FA20	DB 10H　　CONSOLE の最大行数
FA21	DB C3H
FA22 FA23	} LINE で使用
FA24	DB C3H
FA25 FA26	} LINE で使用
FA27	前にシステムポート B0H へ出力したデータ bit 7: ×　bit 6: ×　bit 5: ×　bit 4: ×　bit 3: MOTOR ON,OFF　bit 2–1: V-RAMアドレス切換え　bit 0: TIMER ON,OFF MOTOR: 0… OFF　1… ON V-RAMアドレス切換え: 00…C000H　01…E000H　10…8000H　11…A000H TIMER: 0… ON　1… OFF
FA28 ∫ FA2B	} TIME 用データ
FA2C	プリンタ印字フラグ　　00H：前にプリンタで印字を1回も行なっていない 01H：印字を行なった
FA2D	CONSOLE 第4パラメータ(キーのクリック音発生)用フラグ 00H　　：ON 00H以外：OFF
FA2E	カーソル点滅用フラグ
FA2F	カーソル点滅状態フラグ 00H でノーマルな状態　　FFH で反転の状態にある
FA30	グラフィックキー用ワーク
FA31	グラフィックキー用フラグ
FA32	ファクションキーカウンタ
FA33 ∫ FA46	} ファンクションキー初期データ(第4章参照)

FA33 FA34	COLOR␣
FA35 FA36	CLOAD"
FA37 FA38	GOTO␣
FA39 FA3A	LIST␣
FA3B FA3C	RUN [CR]
FA3D FA3E	SCREEN␣
FA3F FA40	CSAVE"
FA41 FA42	PRINT␣
FA43 FA44	PLAY␣
FA45 FA46	CONT [CR]　　␣=20H(スペースコード)
FA47 ～ FA4B	乱数エリア
FA4C ～ FA50	乱数初期値　0.499766207
FA51 ～ FA55	乱数エリア2　前回の乱数が入っている.
FA57	プリンタ・ヘッド位置　LPOS 用
FA58	出力先 I/O 機器指定 00H：CRT 01H：プリンタ 02H：RS-232C 80H：CMT
FA59	プリント命令のカンマ処理用定数
FA5B FA5C	スタック・ポインタ初期アドレス
FA5D FA5E	BASIC 実行中の行番号

FA5F FA60	BASIC プログラム開始アドレス
FA61 ～ FB3C	BASIC 命令　ジャンプテーブル

アドレス	命令名	中間コード	処理アドレス
FA61,62	END	80	3535
FA63,64	FOR	81	067E
FA65,66	NEXT	82	35F7
FA67,68	DATA	83	07E0
FA69,6A	INPUT	84	09AB
FA6B,6C	DIM	85	3302
FA6D,6E	READ	86	0A09
FA6F,70	LET	87	07F5
FA71,72	GOTO	88	07A0
FA73,74	RUN	89	0781
FA75,76	IF	8A	0861
FA77,78	RESTORE	8B	3519
FA79,7A	GOSUB	8C	078F
FA7B,7C	RETURN	8D	07BC
FA7D,7E	REM	8E	07E2
FA7F,80	STOP	8F	3533
FA81,82	OUT	90	0DD6
FA83,84	ON	91	0844
FA85,86	LPRINT	92	087A
FA87,88	DEF	93	0D3A
FA89,8A	POKE	94	0DFA
FA8B,8C	PRINT	95	087E
FA8D,8E	CONT	96	356B
FA8F,90	LIST	97	05DB
FA91,92	LLIST	98	05D6
FA93,94	CLEAR	99	35A9
FA95,96	COLOR	9A	1D9B
FA97,98	PSET	9B	2D3C
FA99,9A	PRESET	9C	2D37
FA9B,9C	LINE	9D	2DC7
FA9D,9E	PAINT	9E	2EDC
FA9F,A0	SCREEN	9F	1E04
FAA1,A2	CLS	A0	1DF8
FAA3,A4	LOCATE	A1	1CD2
FAA5,A6	CONSOLE	A2	1CF6

FAA7, A8	CLOAD	A3	2496
FAA9, AA	CSAVE	A4	247E
FAAB, AC	EXEC	A5	261D
FAAD, AE	SOUND	A6	1E9B
FAAF, B0	PLAY	A7	1EB3
FAB1, B2	KEY	A8	2353
FAB3, B4	LCOPY	A9	22A6
FAB5, B6	NEW	AA	34CD
FAB7, B8	拡張用 No1	AB	
FAB9, BA	〃 No2	AC	
FABB, BC	〃 No3	AD	
FABD, BE	〃 No4	AE	
FABF, C0	〃 No5	AF	
FAC1, C2	〃 No6	B0	
FAC3, C4	〃 No7	B1	
FAC5, C6	〃 No8	B2	
FAC7, C8	〃 No9	B3	
FAC9, CA	〃 No10	B4	
FACB, CC	〃 No11	B5	
FACD, CE	〃 No12	B6	
FACF, D0	〃 No13	B7	
FAD1, D2	〃 No14	B8	
FAD3, D4	〃 No15	B9	
FAD5, D6	〃 No16	BA	
FAD7, D8	〃 No17	BB	
FAD9, DA	〃 No18	BC	
FADB, DC	〃 No19	BD	
FADD, DE	〃 No20	BE	
FADF, E0	〃 No21	BF	
FAE1, E2	〃 No22	C0	
FAE3, E4	〃 No23	C1	
FAE5, E6	SGN	D4	3898
FAE7, E8	INT	D5	39E7
FAE9, EA	ABS	D6	38B9
FAEB, EC	USR	D7	0755
FAED, EE	FRE	D8	32DE
FAEF, F0	INP	D9	0DCC
FAF1, F2	LPOS	DA	0D22
FAF3, F4	POS	DB	0D27
FAF5, F6	SQR	DC	3F92
FAF7, F8	RND	DD	3BA3
FAF9, FA	LOG	DE	3EA5

FAFB, FC	EXP	DF	3E21
FAFD, FE	COS	E0	3F51
FAFF, 00	SIN	E1	3F57
FB01, 02	TAN	E2	3FD3
FB03, 04	PEEK	E3	0DF3
FB05, 06	LEN	E4	3229
FB07, 08	HEX$	E5	03EA
FB09, 0A	STR$	E6	305B
FB0B, 0C	VAL	E7	32BA
FB0D, 0E	ASC	E8	3238
FB0F, 10	CHR$	E9	3249
FB11, 12	LEFT$	EA	3257
FB13, 14	RIGHT$	EB	3286
FB15, 16	MID$	EC	328F
FB17, 18	POINT	ED	N_{60}内部では使用していない.
FB19, 1A	CSRLIN	EE	
FB1B, 1C	STICK	EF	
FB1D, 1E	STRIG	F0	
FB1F, 20	TIME	F1	
FB21, 22	関数拡張用No1	F2	
FB23, 24	〃 No2	F3	
FB25, 26	〃 No3	F4	
FB27, 28	〃 No4	F5	
FB29, 2A	〃 No5	F6	
FB2B, 2C	〃 No6	F7	
FB2D, 2E	〃 No7	F8	
FB2F, 30	〃 No8	F9	
FB31, 32	〃 No9	FA	
FB33, 34	〃 No10	FB	
FB35, 36	〃 No11	FC	
FB37, 38	〃 No12	FD	
FB39, 3A	〃 No13	FE	
FB3B, 3C	〃 No14	FF	

FB3D ～ FB44	ファンクションキー1　データ　（第4章参照）
FB45 ～ FB4C	ファンクションキー2　データ
FB4D ～ FB54	ファンクションキー3　データ

FB55 ∫ FB5C	ファンクションキー4　データ
FB5D ∫ FB64	ファンクションキー5　データ
FB65 ∫ FB6C	ファンクションキー6　データ
FB6D ∫ FB74	ファンクションキー7　データ
FB75 ∫ FB7C	ファンクションキー8　データ
FB7D ∫ FB84	ファンクションキー9　データ
FB85 ∫ FB8C	ファンクションキー10　データ
FB8D FB8E	ファンクションキー・アドレスポインタ
FB8F ∫ FB94	キー入力用バッファ制御マップ
FB8F FB90 FB91 FB92 FB93 FB94	バッファ内データ先頭の相対位置 バッファ内データ終了の相対位置 フラグ バッファの最大容量　(3FH, 63 バイト) バッファの開始アドレス (*)
FB95 ∫ FB9A	RS-232C 用バッファ制御マップ (内容は(*)と同じ)
FB9B ∫ FBA0	ラウンドロビンバッファ制御マップ　　拡張用 (内容は(*)と同じ)

アドレス	内容
FBA1 ~ FBA6	PLAY チャンネル A 用バッファ制御マップ (内容は(＊)と同じ)
FBA7 ~ FBAC	PLAY チャンネル B 用バッファ制御マップ (内容は(＊)と同じ)
FBAD ~ FBB2	PLAY チャンネル C 用バッファ制御マップ (内容は(＊)と同じ)
FBB3 ~ FBB8	バッファ・フラグ処理用ワーク(詳細は不明)
FBB9 ~ FBF8	キー入力 用バッファ (63バイト)
FBF9 ~ FC38	RS-232C 用バッファ (63バイト)
FC39 ~ FC78	PLAY チャンネル A 用バッファ (63バイト)
FC79 ~ FCB8	PLAY チャンネル B 用バッファ (63バイト)
FCB9 ~ FCF8	PLAY チャンネル C 用バッファ (63バイト)
FCF9 ~ FD1A	PLAY 関係ワーク(詳細は不明)
FD1B	タイマ割り込み内 PLAY 用フラグ 00H で終了
FD1C ~ FD8B	PLAY 各チャンネル用ワークエリア (71H バイト)
FD8C	最大ページ数 01H ~ 04H
FD8D FD8E	BASIC エリアの上限

FD8F	(SCREEN 第2パラ)－1　　アクティブ・ページ
FD90	(SCREEN 第3パラ)－1　　表示ページ
FD91 〜 FDC7	アクティブページ・データ　(**)
FD91	V-RAM(アトリビュートを含む)の開始アドレスの上位(例 80H)
FD92	(SCREEN 第1パラ)－1　　モード
FD93	COLOR 第1パラ　}現在のモードでの COLOR データ
FD94	COLOR 第2パラ
FD95	COLOR 第3パラ
FD96 〜 FD98	モード1用 COLOR データ
FD99 〜 FD9B	モード2用 COLOR データ
FD9C 〜 FD9E	モード3用 COLOR データ
FD9F 〜 FDA1	モード4用 COLOR データ
FDA2	CONSOLE 開始行+1　(1~16)
FDA3	CONSOLE 最終行+1　(1~16)
FDA4	CONSOLE 開始行+1　(1~16)
FDA5	CONSOLE 最終行+1－(FDA6)　(1~16)
FDA6	CONSOLE 第3パラ(ファンクションキー表示) 01H：ON 00H：OFF
FDA7	ファンクションキー表示状態(SHIFT による) 01H：f1~f5 03H：f6~f10
FDA8	カーソル Y 座標+1
FDA9	カーソル X 座標+1
FDAA ADAB	V-RAM 上でのカーソルアドレス
FDAC	カーソル移動可能な X 座標+1の最大値 モード3のとき　10H モード1, 2, 4の時　20H
FDAD	NULL コード　20H
FDAE FDAF	グラフィック命令での X 座標

FDB0 FDB1	}グラフィック命令での Y 座標
FDB2	COLOR データ (例)03H：モード1，20H：モード2，C0H：モード3，80H：モード4
FDB3 FDB4	}CRT アドレス
FDB5 FDB6	}不明　(DW 1010H)
FDB7 ~ FDC6	0～15行までの行フラグ スクリーンエディット用
FDC7	不明
FDC8 ~ FDFE	ページ1用データエリア (FD91H～FDC7H までのデータが，コピーされる.)
FDFF ~ FE35	ページ2用データエリア
FE36 ~ FE6C	ページ3用データエリア
FE6D ~ FEA3	ページ4用データエリア
FEA4 FEA5	}コマンド入力時のカーソル相対位置
FEA6	コマンド入力時の X 座標最大値+1
FEA7	不明
FEA8	INS フラグ　FFH：INS 中　00H：INS 中ではない.
FEA9	LF の時のフラグ
FEAA	STOP した時の(SCREEN 第2パラメータ)－1
FEAB	STOP した時の(SCREEN 第3パラメータ)－1
FEAC	アトリビュートの DATA
FEAD FEAE	}グラフィック X 座標のコピー
FEAF FEB0	}グラフィック Y 座標のコピー

FEB1 FEB2	グラフィック関係ワーク
FEB3 FEB4	LINE 用ワーク
FEB5 ~ FEC5	PAINT 用ワーク
FEC6	不明
FEC7	ファンクションキー表示で表示できる文字数
FEC8	ファンクション表示モード
FEC9	ファンクションのカウンタ
FECA	SPACE, カーソル, STOP, SHIFT 用キーフラグ bit 7: スペース, 6: 00H, 5: ←, 4: →, 3: ↓, 2: ↑, 1: STOP, 0: SHIFT 押されたキーのビットが1になる.
FECB ~ FED0	CMT ファイルネーム バッファ1(キー入力より)
FED1 ~ FED6	CMT ファイルネーム バッファ2(CMT より)
FED7	CMT マークチェックフラグ
FED8	LOAD, VERIFY フラグ 00H : LOAD　エラーすると NEW がかかる. FFH : VERIFY
FED9	キーバッファ用 00H 定数
FEDA ~ FF21	キーバッファ　72文字
FF22 FF23	不明
FF24	DIM フラグ　AFH : DIM　00H : 変数解析
FF25	型フラグ　00H : 数値型　01H : 文字型

FF26	ダブルクォーテーションまたは REM・DATA 出現フラグ
FF27 FF28	BASIC エリア終アドレス
FF29 FF2A	BASIC エリア開始アドレス
FF2B FF2C	文字変数のポインタ
FF2D ∫ FF38	不明
FF39 FF3A FF3B FF3C	文字変数データ 文字数 グミー ポインタ
FF3D FF3E	文字列フリーエリア先頭アドレス
FF3F FF40	汎用ワーク
FF41 ∫ FF44	不明
FF45 FF46	DATA 文　行番号
FF47	FN フラグ
FF48	047AH で使用　詳細は不明
FF49	READ, INPUT フラグ 00H　　：INPUT 00H以外：READ
FF4A ∫ FF4D	不明
FF4E FF4F	STOP, END の時の TEXT アドレス (LET で変数領域ポインタ用ワークとしても使用)
FF50 FF51	汎用ワーク

アドレス	内容
FF52 FF53	}STOP 時の行番号.
FF54 FF55	}次に実行する TEXT のアドレス
FF56 FF57	}変数領域の開始アドレス
FF58 FF59	}配列領域の開始アドレス
FF5A FF5B	}フリーエリアの開始アドレス
FF5C FF5D	}DATA 文のポインタ
FF5E ≀ FF64	}FN で使用
FF65	ダミー
FF66 FF67 FF68 FF69 FF6A	}仮数部（FF66〜FF69） }指数部（FF6A） }FAC①(文字列データの格納場所としても使用)
FF6B	補正用フラグ　80H：正　7FH：負　FFH：整数
FF6C	ダミー
FF6D ≀ FF71	}FAC②
FF72 ≀ FF76 FF77 FF78 ≀ FF7B	}FAC③（FF72〜FF76） }FAC④（FF77〜FF7B） }10進変換で使用
FF7C ≀ FF81	}不明

FF82 ～ FF86	乱数エリア
FF87 ～ FF89	不明
FF8A ～ FFE3	フックエリア
FF8A ～ FF8C	0955H よりフック PRINT#, INPUT# 終了処理から
FF8D ～ FF8F	0404H よりフック エラー処理から
FF90 ～ FF92	0415H よりフック エラー処理から
FF93 ～ FF95	0442H よりフック TEXT EDIT から
FF96 ～ FF98	0472H よりフック TEXT EDIT から
FF99 ～ FF9B	04C4H よりフック TEXT EDIT から
FF9C ～ FF9E	04CAH よりフック TEXT EDIT から
FF9F ～ FFA1	0578H よりフック 中間言語変換から
FFA2 ～ FFA4	0665H よりフック 中間言語逆変換から
FFA5 ～ FFA7	0716H よりフック 実行 MAIN ルーチンから
FFA8 ～ FFAA	0781H よりフック RUN から

FFAB ∫ FFAD	0844H よりフック ON から
FFAE ∫ FFB0	087EH よりフック PRINT から
FFB1 ∫ FFB3	08D2H よりフック PRINT から
FFB4 ∫ FFB6	098BH よりフック INPUT, READ エラー処理から
FFB7 ∫ FFB9	0995H よりフック INPUT, READ エラー処理から
FFBA ∫ FFBC	09B1H よりフック INPUT から
FFBD ∫ FFBF	0A46H よりフック READ, INPUT 共有ルーチンから
FFC0 ∫ FFC2	0B81H よりフック 式解析ルーチンから
FFC3 ∫ FFC5	0C3BH よりフック 式解析，関数処理から
FFC6 ∫ FFC8	2DC7H よりフック LINE から
FFC9 ∫ FFCB	34E0H よりフック NEW から
FFCC ∫ FFCE	34CEH よりフック NEW から
FFCF ∫ FFD1	26C7H よりフック 1文字出力ルーチンから
FFD2 ∫ FFD4	22A6H よりフック LCOPY から

FFD5 ∫ FFD7	1951H よりフック カラー処理関係から
FFD8 ∫ FFDA	0F31H よりフック キー割り込みから
FFDB ∫ FFDD	RST18H で使用
FFDE ∫ FEE0	現在　未使用
FFE1 ∫ FFE3	RST38Hで使用
FFE4 ∫ FFFF	未使用

付-4　中間言語と処理ルーチン対応表

キーワード	中間言語コード	処理ルーチン
ABS	D6	38B9
AND	CF	0C9A
ASC	E8	3238
CHR$	E9	3249
CLEAR	99	35A9
CLOAD	A3	2496
CLS	A0	1DF8
COLOR	9A	1D9B
CONSOLE	A2	1CF6
CONT	96	356B
COS	E0	3F51
CSAVE	A4	247E
CSRLIN	EE	0D30
DATA	83	07E0
DEF	93	0D3A
DIM	85	3302
END	80	3535
EXEC	A5	261D
EXP	DF	3E21
FN	C4	0D61
FOR	81	067E
FRE	D8	32DE
GOSUB	8C	078F
GOTO	88	07A0
HEX$	E5	03EA
IF	8A	0861
INKEY$	C6	2771
INP	D9	0DCC
INPUT	84	09AB
INT	D5	39E7
KEY	A8	2353
LCOPY	A9	22A6
LEFT$	EA	3257
LEN	E4	3229
LET	87	07F5
LINE	9D	2DC7
LIST	97	05DB
LLIST	98	05D6
LOCATE	A1	1CD2
LOG	DE	3EA5
LPOS	DA	0D22
LPRINT	92	087A
MID$	EC	328F
NEW	AA	34CD
NEXT	82	35F7
NOT	C8	0CF9
ON	91	0844
OR	D0	0C99
OUT	90	0DD6

PAINT	9 E	2 E D C
PEEK	E 3	0 D F 3
PLAY	A 7	1 E B 3
POINT	E D	2 D 5 5
POKE	9 4	0 D F A
POS	D B	0 D 2 7
PRESET	9 C	2 D 3 7
PRINT	9 5	0 8 7 E
PSET	9 B	2 D 3 C
READ	8 6	0 A 0 9
REM	8 E	0 7 E 2
RESTORE	8 B	3 5 1 9
RETURN	8 D	0 7 B C
RIGHT$	E B	3 2 8 6
RND	D D	3 B A 3
RUN	8 9	0 7 8 1
SCREEN	9 F	1 E 0 4
SGN	D 4	3 8 9 8
SIN	E 1	3 F 5 7
SOUND	A 6	1 E 9 B
SPC	C 5	0 9 2 6
SQR	D C	3 F 9 2
STICK	E F	2 2 3 6
STEP	C 9	—
STOP	8 F	3 5 3 3
STR$	E 6	3 0 5 B

STRIG	F 0	2 2 5 6
TAB	C 2	0 9 2 6
TAN	E 2	3 F D 3
THEN	C 7	—
TIME	F 1	1 E 8 3
TO	C 3	—
USR	D 7	0 7 5 5
VAL	E 7	3 2 B A
+	C A	3 6 7 E
−	C B	3 6 8 3
*	C C	3 7 B 4
/	C D	3 8 0 3
^	C E	3 E F A
>	D 1	—
=	D 2	—
<	D 3	—

付-5　キャラクタ・コード表

下位4ビット ＼ 上位4ビット	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0		π		0	@	P		p	♠			ー	タ	ミ	た	み
1	月	┴	!	1	A	Q	a	q	♥	あ	。	ア	チ	ム	ち	む
2	火	┬	"	2	B	R	b	r	♣	い	「	イ	ツ	メ	つ	め
3	水	┤	#	3	C	S	c	s	♦	う	」	ウ	テ	モ	て	も
4	木	├	$	4	D	T	d	t	○	え	、	エ	ト	ヤ	と	や
5	金	┼	%	5	E	U	e	u	●	お	・	オ	ナ	ユ	な	ゆ
6	土	│	&	6	F	V	f	v	を	か	ヲ	カ	ニ	ヨ	に	よ
7	日	─	'	7	G	W	g	w	ぁ	き	ァ	キ	ヌ	ラ	ぬ	ら
8	年	┌	(	8	H	X	h	x	ぃ	く	ィ	ク	ネ	リ	ね	り
9	円	┐	)	9	I	Y	i	y	ぅ	け	ゥ	ケ	ノ	ル	の	る
A	時	└	*	:	J	Z	j	z	ぇ	こ	ェ	コ	ハ	レ	は	れ
B	分	┘	+	;	K	[	k	{	ぉ	さ	ォ	サ	ヒ	ロ	ひ	ろ
C	秒	╳	,	<	L	¥	l	¦	ゃ	し	ャ	シ	フ	ワ	ふ	わ
D	百	大	-	=	M	]	m	}	ゅ	す	ュ	ス	ヘ	ン	へ	ん
E	千	中	.	>	N	^	n	~	ょ	せ	ョ	セ	ホ	゛	ほ	
F	万	小	/	?	O	_	o		っ	そ	ッ	ソ	マ	゜	ま	

付-6　キーボード配列表(1)

付-6 キーボード配列表(2)

付-7 エラーメッセージ一覧表

エラーコード	エラー	エラー メッセージ	意　味
0	NF	Next without For Error	NEXTが多すぎる．
2	SN	Syntax Error	文法がまちがっている．
4	RG	Return without Gosub Error	RETURNだけがある．
6	OD	Out of Data Error	DATAがありません．
8	FC	Function Call Error	規定範囲外の数を使った．
10	OV	Over flow Error	計算結果が大き過ぎる．または小さ過ぎる．
12	OM	Out of Memory Error	メモリ容量が足りません．
14	UL	Undefined Line Error	指定した行番号がありません．
16	BS	Bad Subscript Error	配列変数の添字が規定の範囲を超えている．
18	DD	Dupulicate Definition Error	同じ配列を2度定義した．
20	／0	Division by Zero Error	0で割算を行なった．
22	ID	Illegal Direct Error	INPUT, DEFFNをダイレクトモードで行なった．
24	TM	Type Mismatch Error	式の左右の型が一致していない．
26	OS	Out of String Space Error	文字列変数のエリアが足りない．
28	LS	Long String Error	文字列の長さが255文字を超えた．
30	ST	String too Complex Error	文字式が複雑すぎる．
32	CN	Continue Error	CONTはできません．
34	UF	Undefined Function Error	DEFFNで定義されていません．
36	TR	Tape Read Error	テープの読みこみが正しくない．
38	MO	Missing Operand Error	パラメータの指定が不完全．
40	FD	File Data Error	ファイルのデータ形式がまちがっている．

付-8 タイニー・モニタ

本書で解説されている機械語プログラムのキーインやメモリダンプを行なうためのツールとして"TINY MONITOR"プログラムをBASICで作成しましたので，ここにそのリストと使い方を紹介します．

〈プログラム・リスト〉

```
 10 REM TINY MONITOR
 20 CLS:PRINT " <<<TINY MONITOR >>>"
 30 PRINT "S ---  SET MEMORY"
 40 PRINT "D ---  DUMP MEMORY"
 50 INPUT A$
 60 IF LEFT$(A$,1)="s" OR LEFT$(A$,1)="S" THEN 1
    00
 70 IF LEFT$(A$,1)="d" OR LEFT$(A$,1)="D" THEN 3
    00
 80 PRINT :GOTO 50
100 REM set memory
110 A$=RIGHT$(A$,LEN(A$)-1)
120 AD=VAL("&h"+A$)
125 IF AD<0 THEN AD=2^16+AD
130 FOR I=AD TO 65535 STEP4:GOSUB 400:A1$="":FOR
     J=0 TO 3:A=PEEK(I+J)
140 GOSUB 500:PRINT "-";
150 A$=INKEY$:IF A$=""GOTO 150
160 IF A$=" "THEN PRINT "   ";:GOTO 290
170 IF VAL("&h"+A$)=0 AND A$<>"0"THEN PRINT :GOT
    O 50
180 A1$=A$:PRINT A$;
190 A$=INKEY$:IF A$=""GOTO 190
200 IF A$=" "THEN PRINT "  ";:GOTO 290
210 IF VAL("&h"+A$)=0 AND A$<>"0"THEN PRINT :GOT
    O 50
220 A1$=A1$+A$:PRINT A$;" ";:A=VAL("&h"+A1$):POK
    E I+J,A:GOTO 290
290 NEXT J:PRINT:NEXT I:PRINT :GOTO 50
300 REM DUMP MEMORY
310 A$=RIGHT$(A$,LEN(A$)-1)
320 AD=VAL("&h"+A$)
325 IF AD<0 THEN AD=2^16+AD
330 FOR I=AD TO 65535 STEP8:GOSUB 400:A1$="":FOR
     J=0 TO 7:A=PEEK(I+J)
340 GOSUB 500:PRINT " ";
350 A$=INKEY$:IF A$<>""THEN PRINT :GOTO 50
360 NEXT J:PRINT :NEXT I:PRINT :GOTO 50
400 A=INT(I/256):GOSUB 500:A=I-(INT(I/256)*256):
    GOSUB 500
410 PRINT " ";:RETURN
500 A1=INT(A/16):A2=A-(A1*16)
510 IF A1>=10 THEN A1=A1+7
520 A1=A1+&H30
530 IF A2>=10 THEN A2=A2+7
540 A2=A2+&H30
550 PRINT CHR$(A1)+CHR$(A2);:RETURN
```

《使い方》

プログラムをキーインして，RUN させると，次のように表示され，入力待ちになります.

```
  〈TINY MONITOR〉
S ……… SET MEMORY
D ……… DUMP MEMORY
?
```

機械語を書き込む場合は S を，メモリーダンプを行なうには D を選びますが，S やD の後に続けて，アドレスを16進で指定する必要があります.

```
ex)  S C000
     d BF00
```

書き込みの際，アドレスに既に書き込まれている値が表示されますが，これを変更する必要がない場合，スペースバーを押すと，次のアドレスの入力に移ります.

S または D のモードから抜けだすには，16進キー(0～F)以外のキーを押します.

付-9　12平均率音階表

	1	2	3	4	5	6	7	DATA
C	32.70	65.41	130.81	261.63	523.25	1046.50	2093.01	32.70
C＃	34.65	69.30	138.59	277.18	554.37	1108.73	2217.46	34.65
D	36.71	73.42	146.83	293.67	587.33	1174.66	2349.02	36.70
D＃	38.89	77.78	155.56	311.13	622.25	1244.51	2489.02	38.89
E	41.20	82.41	164.81	329.63	659.26	1318.51	2637.02	41.20
F	43.65	87.31	174.61	349.23	698.46	1396.91	2793.83	43.65
F＃	46.25	92.50	185.00	370.00	739.99	1474.98	2959.96	46.25
G	49.00	98.00	196.00	392.00	783.99	1567.98	3135.96	49.00
G＃	51.91	103.83	207.65	415.31	830.61	1661.22	3322.44	51.91
A	55.00	110.00	220.00	440.00	880.00	1760.00	3520.00	55.00
A＃	58.27	116.54	233.08	466.16	932.33	1864.66	3729.31	58.26
B	61.74	123.47	246.94	493.88	987.77	1975.53	3951.07	61.72

付-10　サウンドレジスター覧表

レジスタ ＼ ビット		B 7 MSB	B 6	B 5	B 4	B 3	B 2	B 1	B 0 LSB
R_0	チャネルA音階	8bit Fine Tune A							
R_1						4bit Coarse Tune A			
R_2	チャネルB音階	8bit Fine Tune B							
R_3						4bit Coarse Tune B			
R_4	チャネルC音階	8bit Fine Tune C							
R_5						4bit Coarse Tune C			
R_6	ノイズ周波数				5bit Period Control				
R_7	イネーブル	IN/OUT		ノイズ			トーン		
		I/O B	I/O A	C	B	A	C	B	A
R_8	チャネルA音量				M	L_3	L_2	L_1	L_0
R_9	チャネルB音量				M	L_3	L_2	L_1	L_0
R_{10}	チャネルC音量				M	L_3	L_2	L_1	L_0
R_{11}	エンベロープ周期	8bit Fine Tune E							
R_{12}		8bit Coarse Tune E							
R_{13}	エンベロープ形状					CONT.	ATT.	ALT.	HOLD
R_{14}	I/Oポート A データ・ストア	8bit パラレルI/O　ポート A							
R_{15}	I/Oポート B データ・ストア	8bit パラレルI/O　ポート B							

索　引

〈著者代表略歴〉

一　零（かずと　れい）

昭和30年8月1日　博多生まれ
昭和51年　九州産業大学電気工学科卒業
現在株式会社システムソフト福岡技術顧問
パーソナルコンピュータのハードウェア・ソフトウェアの両方に精通し，特に機械語でのプログラム開発に特異性を発揮，"頭の中"でアセンブル・逆アセンブルを行なうため，さすらいのアセンブラーと呼ばれている．現在，史上最大のパソコン・ウォーゲームを開発中．

本書の内容に関しまして，ご質問・ご意見がございましたら下記の出版部まで書面にてお寄せくださいますようお願い致します．

PCファミリー・テクニカル・ノウハウ集
PC-6000シリーズ編
PC-Techknow 6000 Vol. 1

共　著	一　零・樋口　理 八木　良一
監　修	システムソフト
発行者	塚本　慶一郎

発行所　株式会社　アスキー出版

〒150　東京都渋谷区神宮前5－2－2　瀬川ビル
振替　東京7－57496
☎03-407-4910(業務部)　03-407-4903(出版部)

1982年9月30日　第1版第1刷発行　Printed in Japan
2,500円

ISBN4-87148-287-1　C3055　¥2500E